НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«До захисту допущено»
Науковий керівник кафедри
_____ І.А. Дичка
«___»_____2019 р.

Дипломний проект

на здобуття ступеня бакалавра

з напряму підготовки 6.050103 «Програмна інженерія»

на тему: «Програмна платформа для автоматизації надання послуг рекламним компаніям»

Виконав:
студент IV курсу, групи КП-51
Кривенко Петро Олегович _____

Керівник:
Ст. викл. кафедри ПЗКС,
Гадиняк Р.А. _____

Консультант з нормоконтролю:
Доцент кафедри ПЗКС, к.т.н.,
Онай М.В. _____

Рецензент:
Доцент кафедри ММСА ІПСА, к.т.н.,
Дідковська М.В. _____

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2019 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет прикладної математики
Кафедра програмного забезпечення комп'ютерних систем

Рівень вищої освіти – перший (бакалаврський)

Напрям підготовки – 6.050103 «Програмна інженерія»

ЗАТВЕРДЖУЮ
Науковий керівник кафедри
_____ І.А. Дичка
«___»_____ 2018 р.

ЗАВДАННЯ

на дипломний проект студенту

Кривенку Петру Олеговичу

1. Тема проекту «Програмна платформа для автоматизації надання послуг рекламним компаніям», керівник проекту Гадиняк Руслан Анатолійович, старший викладач, затверджені наказом по університету від «22» травня 2019 року № 1331-С.

2. Термін подання студентом проекту «20» червня 2019 р.

3. Вихідні дані для дипломного проектування: див. Технічне завдання.

4. Зміст проекту:
   - провести аналіз існуючих аналогів та розглянуті основні методи тестування програмного забезпечення;
   - сформувати вимоги до користувацького інтерфейсу програми;
   - розробити базу даних для збереження даних рекламних компаній;
   - виконати програмну реалізацію платформи відповідно до вимог технічного завдання;
   - виконати тестування платформи.

5. Перелік обов'язкового ілюстративного матеріалу:
   - структура бази даних (плакат);
   - діаграма прецедентів (креслення);
   - алгоритм генерації варіацій (креслення);
   - структурна схема системи (плакат).

6. Консультанти розділів проекту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Онай М.В., доцент кафедри ПЗКС, к.т.н.		

7. Дата видачі завдання «31» жовтня 2018 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломного проекту	Термін виконання етапів проекту	Примітка
1.	Вивчення літератури за тематикою проекту	14.11.2018	
2.	Розробка та узгодження технічного завдання	15.12.2018	
3.	Підготовка матеріалів першого розділу дипломного проекту	30.12.2018	
4.	Розроблення структури платформи	15.01.2019	
5.	Розроблення дизайну сторінок та графічних елементів	03.02.2019	
6.	Підготовка матеріалів другого розділу дипломного проекту	01.03.2019	
7.	Програмна реалізація і тестування платформи	12.04.2019	
8.	Підготовка третього розділу дипломного проекту	25.04.2019	
9.	Підготовка четвертого розділу дипломного проекту	02.05.2019	
10.	Підготовка графічної частини дипломного проекту	19.05.2019	
11.	Оформлення документації дипломного проекту	26.05.2019	

Студент _____ Кривенко П.О.

Керівник проекту _____ Гадиняк Р.А.

АНОТАЦІЯ

Дана робота присвячена розробленню платформи для автоматизації надання послуг рекламним компаніям.

У роботі виконано порівняльний аналіз існуючих рішень для створення та поширення рекламного контенту, методів генерації рекламних креативів та можливостей їх автоматизації. Розроблена платформа налає рекламодавцям можливості створювати різноманітну рекламну продукцію, враховуючи такі властивості аудиторії, як місцезнаходження, стать, вік тощо, а також автоматично розміщувати її у мережі веб. Система надає засоби моніторингу перебігу рекламної кампанії, а саме можливість відстеження основних показників активності аудиторій, на основі чого можна робити висновки про успішність кампанії та використовувати ці дані для подальшого покращення. Результатом роботи платформи є повна автоматизація рекламної кампанії, від обрання тип рекламного креативу, генерації зображень різних розмірів та наповнення, розміщення реклами на різних сервісах, до аналізу статистики.

У даному дипломному проекті розроблено: архітектуру серверної та клієнтської частини web-сервісу, алгоритми генерації даних, алгоритми публікації рекламних креативів для різних платформ, алгоритм збору та аналізу статистики, а також графічні елементи та дизайн web-сторінок.

ABSTRACT

This work is devoted to the development of a platform for automating the provision of services to advertising companies.

In the work the comparative analysis of existing solutions for creation and distribution of advertising content, methods of generation of advertising creatives and possibilities of their automation is made. The developed platform puts advertisers the opportunity to create a variety of advertisement inventory, taking into account such audience features as location, gender, age, and more, and automatically post it on the web.

The system provides means for monitoring the progress of an advertising campaign, namely the ability to track key performance indicators for audiences, on the basis of which it is possible to draw conclusions about the success of the campaign and use this data for further improvement. The result of the platform's operation is the complete automation of the advertising campaign, from the selection of the type of advertising creative, the generation of images of various sizes and content, advertising on various services, to the analysis of statistics.

This graduation project has developed: the architecture of the server and client part of the web-service, algorithms for data generation, algorithms for publishing advertising creatives on various platforms, algorithm for collecting and analyzing statistics, as well as graphic elements and web-page design.

ДП.045440-01-90 Програмна платформа для автоматизації надання послуг рекламним компаніям. Відомість проекту

Позначення	Найменування	Кіл-ть	Примітка
	Документація проекту		
ДП.045440-02-91	Програмна платформа для	5	
	автоматизації надання		
	послуг рекламним		
	компаніям. Технічне		
	завдання		
ДП.045440-03-81	Програмна платформа для	62	
	автоматизації надання		
	послуг рекламним		
	компаніям. Пояснювальна		
	записка		
ДП.045440-04-51	Програмна платформа для	4	
	автоматизації надання		
	послуг рекламним		
	компаніям. Програма та		
	методика тестування		
ДП.045440-05-34	Програмна платформа для	13	
	автоматизації надання		
	послуг рекламним		
	компаніям. Керівництво		
	користувача		

Позначення	Найменування	Кіл-ть	Примітка
ДП.045440-06-99	Програмна платформа для	1	
	автоматизації надання		
	послуг рекламним		
	компаніям. Алгоритм		
	генерації варіацій.		
	Блок-схема		
ДП.045440-07-99	Програмна платформа для	1	
	автоматизації надання		
	послуг рекламним		
	компаніям. Діаграма		
	прецедентів		
ДП.045440-08-98	Програмна платформа для	1	
	автоматизації надання		
	послуг рекламним		
	компаніям. Компакт-диск		

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

"ЗАТВЕРДЖЕНО"

Науковий керівник кафедри

_____ І.А. Дичка

"___" _____ 2018 р.

ВЕБ-ДОДАТОК ДЛЯ АВТОМАТИЗАЦІЇ ОФЛАЙН-ПЕРЕВІРКИ ПИСЬМОВИХ ТЕСТІВ

Технічне завдання

ДП.045440-02-91

"ПОГОДЖЕНО"

Керівник проекту:

_____ Р.А.Гадиняк

Нормоконтроль:

_____ М.В. Онай

Виконавець:

_____ П.О. Кривенко

2018

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: Програмна платформа для автоматизації надання послуг рекламним компаніям.

Галузь застосування: інформаційні технології.

2. ПІДСТАВА ДЛЯ РОЗРОБЛЕННЯ

Підставою для розроблення є завдання на дипломне проектування, затверджене кафедрою програмного забезпечення комп'ютерних систем Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського» (КПІ ім. Ігоря Сікорського).

3. ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для використання рекламними компаніями та агентствами в інструменту для автоматизації ведення рекламних кампаній.

4. ВИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ

Платформа повинна забезпечувати такі основні функції:

1. Система передбачає авторизацію користувачів для роботи з нею
2. Система повинна мати дві групи користувацьких ролей, бізнес ролі та адміністративні ролі.
3. Система повинна мати три типи бізнес користувачів: партнер, бренд, рекламодавець.
4. Система повинна мати два типи адміністративних користувачів: адміністратор, головний адміністратор.
5. Створення бріфу.
6. Створення рекламного акаунту.

7. Створення рекламної кампанії.
8. Створення аудиторій.
9. Створення рекламних креативів.
10. Генерація рекламних креативів.
11. Підтвердження рекламних креативів.
12. Публікація рекламних креативів.

Розробку виконати мовою JavaScript.

Додаткові нефункціональні вимоги:

1) динамічна зміна рекламного креативу графіку при зміні параметрів;
2) мінімалістичний дизайн сторінок інтерфейсу;
3) наявність повідомлень про помилки у випадку некоректно введених користувачем даних;
4) наявність спливаючих повідомлень з важливою інформацією.

5. ВИМОГИ ДО ПРОЕКТНОЇ ДОКУМЕНТАЦІЇ

У процесі виконання проекту повинна бути розроблена наступна документація:

1) пояснювальна записка;
2) програма та методика тестування;
3) керівництво користувача;
4) креслення:
    - Діаграма прецедентів.
    - Алгоритм генерації варіацій. Діаграма діяльності.

6. ЕТАПИ ПРОЕКТУВАННЯ

Вивчення літератури за тематикою роботи……..……..30.11.2018

Розроблення та узгодження технічного завдання…..……...15.12.2018

Розроблення структури веб-додатку...…….…….…….....03.01.2019

Розроблення алгоритму розпізнавання……..…....10.01.2019

Програмна реалізація веб-додатку…….....10.02.2019

Створення інтерфейсу...…….....10.03.2019

Тестування веб-додатку...…….....03.04.2019

Підготовка матеріалів текстової частини проекту……18.04.2019

Підготовка матеріалів графічної частини проекту……..…12.05.2019

Оформлення технічної документації проекту…..……..…25.05.2019

7. ПОРЯДОК ТЕСТУВАННЯ РОЗРОБКИ

Тестування розробленого програмного продукту виконується відповідно до “Програми та методики тестування”.

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

"ЗАТВЕРДЖЕНО"

Науковий керівник кафедри

_____ І.А.Дичка

"___" _____ 2019 р.

ПРОГРАМНА ПЛАТФОРМА ДЛЯ АВТОМАТИЗАЦІЇ НАДАННЯ ПОСЛУГ РЕКЛАМНИМ КОМПАНІЯМ

Пояснювальна записка

ДП.045440-03-81

"ПОГОДЖЕНО"

Керівник проекту:

_____ Р.А. Гадиняк

Нормоконтроль:

_____ М.В. Онай

Виконавець:

_____

П.О.Кривенко

2019

ЗМІСТ

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

HTTP (Hyper Text Transfer Protocol) – це протокол клієнт-серверної взаємодії, що означає ініціювання запитів до сервера самим одержувачем, зазвичай веб-браузером.

AMQP (Advanced Message Queuing Protocol) – протокол черги повідомлень, стандартний протокол кадрування та передачі даних, що забезпечує асинхронну, надійну та безпечну передачу повідомлень між двома сторонами.

СКБД (Система керування базами даних) – набір лінгвістичних та програмних засобів спеціального чи загального значення, що забезпечують керування створенням та використанням баз даних.

Конверсія – відношення числа користувачів, що виконали дію, зазначену рекламодавцем, після перегляду реклами, до загального числа користувачів, що її переглянуло.

JSON (JavaScript Object Notation) – текстовий формат обміну даними, що заснований на JavaScript та складається із об’єктів даних, представлених у вигляді ключ-значення.

BSON (Binary JSON) – формат електронного обміну цифровими даними, заснований на JavaScript, що має бінарну форму представлення простих структур даних.

HTML (Hyper Text Markup Language) – мова розмітки документів, створених для відображення у браузері.

CSS (Cascading Style Sheets) – формальна мова опису зовнішнього вигляду документа, написаного за допомогою мови розмітки HTML.

SASS (Syntactically Awesome Stylesheets) – це метамова сценаріїв, що інтерпретується до каскадної таблиці стилів (CSS), призначена для спрощення файлів каскадних таблиць стилів та збільшення рівня абстракції кода.

V8 – це високо-продуктивний JavaScript рушій, з відкритим кодом, що написаний на мові C++.

SOLID – акронім для п’яти основних принципів об’єктно-орієнтованого проектування та програмування.

API (Application Programming Interface) – набір засобів, таких як класи, процедури, функції тощо, завдяки яким одна комп’ютерна програма може взаємодіяти з іншою.

JWT (JSON WEB TOCKEN) – це відкритий стандарт для створення токенів доступу, що заснований на стандарті JSON.

CRUD – акронім, що визначає чотири основні функції, які використовуються при роботі з базами даних, а саме «створення, читання, оновлення і видалення».

JPEG (Joint Photographic Experts Group)– один з растрових графічних форматів, що застосовується для зберігання фотографій та зображень.

PNG (Portable Network Graphic) – растровий формат зберігання графічної інформації, що використовує алгоритм стискання без втрат.

REST(Representational State Transfer) – це архітектурний стиль взаємодії компонентів розподіленої системи у мережі. Представляє собою погоджений набір обмежень, що враховуються при проектуванні розподіленої системи.

Інкапсуляція – це принцип об’єктно-орієнтовного програмування, властивість системи, що дозволяє об’єднати дані та методи, що працюють з ними у класі та приховати деталі реалізації від користувача.

Аудиторія – це кількість читачів, слухачів чи глядачів певного носія реклами.

Бріф –це короткий опис, що містить необхідну інформацію яку слід використовувати під час створення рекламної кампанії та креативів.

OLAP (Online Analytical Pprocessing) – це технологія обробки даних, що полягає у підготовці агрегованої інформації на основі великих масивів даних, що структуровані за багатомірним принципом.

Варіація – це креатив з набору рекламних креативів, що відрізняються між собою лише незначними деталями. Це робиться як правило для того, щоб краще пристосуватись під конкретну аудиторію чи медіа-майданчик.

Оркестрування – це автоматичне розміщення, координація та управління складними комп'ютерними системами та службами. Описує те, як мікросервіси мають взаємодіяти між собою, використовуючи для цього обмін повідомленнями, включаючи бізнес-логіку та чітку послідовність дій.

ВСТУП

З кожним днем кількість користувачів мережі Інтернет зростає з небаченою раніше швидкістю. Сьогодні люди користуються Інтернетом у різноманітних сферах життя, таких як освіта, спілкування та бізнес. Через таку популярність рекламний бізнес також вимушений мігрувати до Інтернету, адже це надає йому безпрецедентну можливість: максимально влучно знаходити аудиторію для своєї реклами. Щодня люди використовують різні пошукові системи, соціальні мережі, сервіси для прослуховування музики та перегляду відео. Кожен з таких сервісів має свою аудиторію, яку намагається досконально вивчити. Дані про користувачів постійно збираються такими сервісами для подальшого вивчення та передбачення поведінки своєї аудиторії. На основі цих даних вони мають можливість поділити їх за усіма відомими їм параметрами, такими як стать, вік, національність, місце розташування, сімейний стан, хобі та вподобання. Завдяки наявності цих даних про користувачів, конверсія Інтернет-реклами є значно вищою ніж у будь-якого іншого виду реклами. Майже кожен сервіс, що має свою аудиторію надає можливість показу реклами для неї. Рекламодавці мають унікальну можливість знайти потенційного споживача на свої товари та послуги за допомогою таких платформ та сервісів. Але сьогодні кількість таких сервісів є неймовірно великою та з кожним днем з’являються все нові і нові платформи. Для охоплення максимальної аудиторії рекламодавцям доводиться використовувати їх паралельно. Кожен такий сервіс має свої певні особливості і специфіку роботи, саме тому рекламодавцям, щоб підтримувати паралельне їх використання необхідна велика кількість ресурсів та часу.

Крім того, разом з підвищенням складності підтримки, збільшився час початкового запуску рекламної кампанії. Кожна нова рекламна кампанія від

початку створення перших макетів до свого запуску, проходить через велику кількість схвалень та заперечень на різних рівнях, а також зазнає багатьох змін. З додаванням кожної нової платформи, час необхідний на запуск такої кампанії дуже сильно зростає. Це відбувається через необхідність зміни рекламних стратегій, креативів та інших параметрів для кожної з таких систем. Усі такі зміни мають пройти через ряд схвалень на різних рівнях починаючи від дизайнерів та закінчуючи директорами брендів. У випадку заперечення на одному з таких етапів цей процес починається з початку.

У цій дипломній роботі пропонується рішення проблеми, що була зазначена вище, а саме створення системи, яка надає можливість автоматизації ініціалізації нових рекламних кампаній, генерації рекламних креативів, їх адаптацію під різні рекламні майданчики, а також автоматизовану публікацію цих креативів до різних рекламних платформ.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ РІШЕНЬ

1.1. Аналіз ринку надання послуг рекламним компаніям

В наші дні, серед великого різноманіття різних типів реклами, Інтернет-реклама вважається одним з найефективніших її видів. Саме вона дозволяє проводити оперативне інформування потенційних споживачів, отримувати максимально швидкий відгук, а також надає можливість точного таргетування і попадання у бажану цільову аудиторію. Реальну ефективність такої реклами дуже легко проаналізувати, що у свою чергу, дозволяє рекламодавцям оперативно скоригувати рекламну стратегію у разі необхідності [1]. Однією з найбільших переваг Інтернет-реклами у порівнянні з багатьма іншими видами є ціна, адже розміщення в Інтернеті обходиться значно дешевше. Розподілити Інтернет-рекламу можна за наступними критеріями:

- *За типом контенту***.** Реклама може бути графічною (різні типи банерів, як анімовані так і статичні), текстовою, а також у форматі аудіо- та відеороликів.
- *За технологією доведення інформації до користувача* Розрізняють статичну рекламу на різних тематичних сайтах (вона може набувати вигляду відеоролику, банеру та рекламної статті), директ-розсилку, за якої відбувається розсилка рекламних повідомлень безпосередньо певній групі користувачів, контекстну рекламу (блоки тексту у видачі пошукового рушія) та ін.
- *За типом рекламного майданчика*. Реклама може бути розміщена у пошуковій видачі, у відеоагрегаторах (наприклад, відеоролик, що показується перед самим відео), а також у соціальних мережах у вигляді комерційних постів у різних спільнотах та блоків реклами.
- *За форматом донесення***.** Розрізняють приховану та нативну рекламу за форматом доведення до споживача. Нативна реклама

зазвичай імітує певний контент майданчика на показується, але у даному випадку завжди зазначено те, що це є рекламним матеріалом. Проте, варто зазначити, що часто ця класифікація є умовною і тому деякі типи реклами одночасно можуть бути розподіленими до кількох таких категорій. Наприклад, пряма реклама може бути опублікованою у соціальних мережах або бути розісланою користувачам, нативна реклама може бути відеорекламою або текстовою і так далі.

Не існує універсального рішення, щодо виду та типу реклами. Вид реклами слід обирати в залежності від маркетингових цілей рекламодавця.

Різновиди Інтернет реклами: текстова, банерна, контекстна, у соціальних мережах/блогах та нативна.

Текстова реклама. Цей вид Інтернет-реклами є досить ефективним. Він представляє собою текстове оголошення, що окрім тексту може містити зображення та є інтегрованим у статтю відповідного змісту. Досить часто такий рекламний блок сприймається як продовження тематичної статті на певному спеціалізованому ресурсі і тому майже завжди споживачі, що цікавляться матеріалом зазначеним у статті сприймають його за продовження.

Банерна реклама. Банери – це невеликі анімовані або графічні блоки, після натискання на які, користувача перенаправляє на сайт рекламодавця. В основному банери розміщуються на різних сайтах і форумах. Банерна реклама є особливо ефективною у випадку, коли потрібно вивести новий продукт на ринок, а також підвищити впізнаваємість бренду серед користувачів обраної аудиторії, адже навіть якщо користувач не перейде за посиланням банеру, то він все-рівно перегляне його контент, а значить, може запам’ятати назву і логотип бренду. Банери також дуже гарно підходять для реклами короткострокових акцій і навіть нових продуктів [2]. Банерна реклама є найстарішим типом реклами у мережі Інтернет та до сих пір є

вельми ефективною, хоча різними сервісами для блокування реклами було завдано значного удару по банерах, особливо на протязі останніх декількох років. Досить часто, один із різновидів банерної реклами – ролики – виділяють в окрему категорію. Зазвичай це картинки з різноманітними інтригуючими заголовками, у яких вказано лише частину інформації про певний продукт чи послугу. Для того, щоб отримати решту інформації користувачеві необхідно перейти по посиланню даного банеру.

Контекстна реклама. Даний тип реклами є мабуть найефективнішим у мережі Інтернет. Вона розміщується на тематичних ресурсах, які є цікавими потенційному споживачеві, у вигляді рекламних оголошень [3]. Її формат та контент максимально точно спрямований на обрану цільову аудиторію. Наприклад, якщо рекламодавець бажає продати певні продукти харчування, то його контекстна реклама буде розміщеною на різноманітних жіночих форумах та кулінарних ресурсах. У випадку, якщо рекламодавець займається організацією туристичних екскурсій та бажає продати путівку на певний курорт, то його рекламу буде показано відвідувачам туристичних блогів і сайтів, що стосуються подорожей і так далі. Такі сервіси як Google Adwords дозволяють розмістити адресу ресурсу у пошуковій видачі, і якщо рекламодавець, наприклад, займаєтеся продажами побутової техніки, то користувач, який задасть пошуковий запит «купити пральну машинку», першим у списку ресурсів побачить ресурс, який було налаштовано. Відповідність реклами тому чи іншому ресурсу, або пошуковому запиту, визначається набором ключових слів, тобто залежить від контексту ресурсу, або пошукового запиту. Тому даний тип Інтернет-реклами має саме таку назву.

Контекстна реклама має безліч переваг, а саме:

1. Точне попадання у цільову аудиторію.

2. Оплата за перехід, тобто рекламодавець платить лише за тих користувачів, які дійсно перейшли на сайт, саме з даного рекламного оголошення.
3. Можливість точного налаштування реклами, щоб її побачила тільки певна група людей.
4. Швидкий результат.
5. Максимальна конверсія і невисока вартість. Споживачі приходять на сайт з подібного за контекстом та тематикою ресурсу, а отже висока ймовірність того, що вони є зацікавленими у товарах та послугах, що надає рекламодавець. Саме тому кожен перехід є потенційною конверсією для рекламодавця.

Однією з головних особливостей контекстової реклами, є той факт, що вона не викликає роздратування у споживачів, оскільки її бачать виключно ті, хто вже зацікавлений в подібних товарах і послугах. Всі ці особливості роблять контекстну рекламу зручним та вигідним інструментом для будь-якого бізнесу

Реклама у соціальних мережах та блогах. Аналогічно до контекстної реклами, реклама у соціальних мережах, таких як Facebook, Twitter та ін., є також вузько таргетованою, тобто вона буде показана лише тим, хто є потенційно зацікавленим у товарі чи послузі. Перевага реклами у соціальних мережах також полягає в тому, що рекламодавець має можливість задавати аудиторію і налаштовувати показ оголошень з великою кількістю параметрів, таких як вік і стать, місце розташування, сімейний стан та інтереси. Така реклама може бути банерною, тоді вона являє собою невеликий блок з заголовком, картинкою, коротким описом і посиланням на ресурс чи сайт.

Нативна реклама є відносно дорогим, але при цьому дуже ефективним видом реклами у мережі Інтернет. Нативна реклама – це оглядова, розважальна, або пізнавальна стаття, що є спеціально написаною

для певного ресурсу, у якій повністю дотримано стилістику цього ресурсу, але при цьому у ній просувається певний бренд, товар, або компанія.

Необхідною умовою для успіху рекламної кампанії є персоналізація рекламного контенту що сприяє кращим продажам товарів та послуг через імітацію спорідненості з користувачем. Зазвичай усі рекламні креативи мають однаковий формат, компонування та переслідують спільну мету, але можуть використовувати дещо різні зображення, заголовки, тексти та інші дані, що є характерними та більш сприятливими для певної аудиторії. Процес адаптації реклами під кожну аудиторію є мануальним та займає багато часу та ресурсів, особливо коли необхідно охопити велику кількість різних аудиторій та досягти максимальної конверсії серед кожної з них.

Рекламний креатив – це мультимедійний об’єкт, який має рекламну функцію та розміщується у рекламних мережах. Такі креативи можуть набувати форм зображень, відео, HTML (великими літерами) шаблонів тощо та є основним інструментом просувань товарів та послуг. Кожна з таких форм в свою чергу має велику кількість форматів, наприклад для HTML банеру існує 14 основних розмірів банер [4], таких як «хмарочос», «квадрат», «білборд» та інші. А, отже, є безліч варіантів представлення своєї реклами кінцевому користувачеві. Для того, щоб успішно прорекламувати свої послуги та отримати максимальну конверсію потрібно врахувати безліч факторів таких як формат рекламного креативу, його контент, обрані канали розповсюдження, вік та стать аудиторії тощо.

Найбільш популярними каналами поширення реклами є такі платформи як Facebook, YouTube та Google завдяки своїй популярності, обсягу та різноманітності аудиторії та можливостям автоматизації процесу видачі реклами кінцевому користувачеві. Зазначені платформи мають спеціальні сервіси для рекламодавців для зручного розміщення та управління рекламним інвентарем. Наприклад, основним сервісом від

Google є Google Campaign Manager також відомий як Google DCM/DFA, який надає наступні можливості для керування рекламою:

- створення та управління плейсментами – внутрішньою сутністю системи яка відповідає за тип, формат, та оплату;
- створення та управління рекламою – внутрішньою сутністю системи яка відповідає за різні типи таргетингів;
- створення та управління сайтами – внутрішньою сутністю системи яка відповідає за місце показу реклами (певний веб-ресурс, мобільний додаток, тощо).

Ці три сутності дозволяють однозначно задати, який рекламний інвентар, на якому ресурсі та якій аудиторії буде показано через заданий канал розповсюдження.

Не зважаючи на зручність управління рекламою у рамках певної з платформ, рекламодавець досить часто зустрічається з проблемою одночасного ведення рекламних кампаній у цих системах, бо всі вони мать різний інтерфейс та різний набір специфічних внутрішніх сутностей, якими рекламодавець повинен оперувати під час проведення рекламної кампанії.

Іншою, не менш суттєвою проблемою для рекламодавця є те, що жодна з платформ не надає можливості для автоматизації адаптації рекламного креативу під різні сценарії використання і кожен з таких креативів має бути завантажений та налаштований вручну для кожного сценарію.

1.2. Аналіз вимог до функціональності програмних засобів

В результаті дослідження сфери рекламних послуг та аналізу ведення рекламних кампаній різними рекламодавцями, можна стверджувати, що проблема з автоматизацією ведення рекламної кампанії є досить актуальною.

В процесі аналізу вимог було сформовано наступні функціональні вимоги:

1. Система повинна мати дві групи користувацьких ролей, бізнес ролі та адміністративні ролі.
2. Система повинна мати три типи бізнес користувачів: партнер, бренд, рекламодавець.
3. Система повинна мати два типи адміністративних користувачів: адміністратор, головний адміністратор.
4. Система передбачає авторизацію користувачів для роботи з нею.
5. Створення бріфу. Бріф – це об’єкт, який визначає основні вимоги та умови, що створені бізнес користувачем. Цей короткий опис містить необхідну інформацію яку слід використовувати під час створення рекламної кампанії та креативів. Бізнес користувач повинен мати можливість крок за кроком надати необхідну інформацію адміністративному користувачеві, таку як назву, дати проведення кампанії, бажані метрики, контакти залучених сторін, основні формати та типи для створення рекламної кампанії.
6. Створення рекламного акаунту. Сутність за допомогою якої рекламна кампанія пов’язується з різними каналами розміщення реклами (Facebook, YouTube, Google).
7. Створення рекламної кампанії. На основі даних, що були задані клієнтом у бріфі, адміністративний користувач створює рекламну кампанію для клієнта та завантажує до кампанії шаблон та динамічні дані для креативів, що були попередньо створені дизайнерами та маркетинговою командою.
8. Створення аудиторій. Аудиторії містять у собі інформацію кому і за яких умов показати рекламний креатив, до якого вони застосовані.

9. Створення рекламних креативів. На основі завантаженого шаблону, оператор має можливість вказати набір значень для кожного динамічного поля у шаблоні, таких як текст, зображення та ін., а також вказати бажану аудиторію для даного набору креативів.
10. Генерація рекламних креативів. На основі шаблону, його статичних полів та динамічних значень заданих оператором та таргетингів генерується набір рекламних креативів.
11. Підтвердження рекламних креативів. Після створення та генерації рекламних креативів адміністративним користувачем, клієнт має можливість підтвердити або заперечити кожний згенерований креатив.
12. Публікація рекламних креативів. Клієнт має можливість опублікувати підтверджені рекламні креативи до вибраних каналів.

2. АНАЛІЗ ТЕХНОЛОГІЙ, АРХІТЕКТУРНИХ ПІДХОДІВ ТА МОВ ПРОГРАМУВАННЯ

2.1. Порівняння монолітної та мікросервісної архітектури.

Архітектура – це базова організація системи, що описує зв’язки між компонентами цієї системи (і зовнішнім середовищем), а також визначає принципи її проектування та розвитку.

Монолітна архітектура означає що ваш додаток – це великий пов'язаний модуль, де всі компоненти спроектовані так, щоб працювати разом один з одним, а також мають спільну пам'ять і ресурси.

Основні переваги монолітної архітектури:

- *Зрозуміла та проста реалізація.* Монолітна архітектура є досить простою у реалізації, управлінні та розгортанні системи. Для реалізації та підтримки монолітної системи можна використовувати розробників середньої групи кваліфікації.
- *Узгодженність.* При монолітній архітектурі простіше підтримувати узгодженість коду, обробляти помилки і т. д. А також завдяки внутрішньо-процесній взаємодії легко забезпечити транзакційну цілісність даних.
- *Міжмодульний рефакторинг.* Єдина архітектура полегшує роботу в ситуаціях, коли кілька модулів повинні взаємодіяти між собою або коли ми хочемо перемістити класи з одного модуля в інший. У випадку з мікросервісами ми повинні дуже чітко визначати межі модулів. Наприклад, багато додатків зазвичай використовують одну і ту ж саму типову функціональність: логування, права доступу і т.д.

Недоліки:

- Необхідність використання єдиного технологічного стеку для всієї системи.

- Велика зв’язність між компонентами системи (при зміні чи рефакторингу однієї частини коду потрібно змінювати усі пов’язанні частини).
- Необхідність перезпуску усієї системи при появі нової функціональності.
- Для ведення проекту необхідне знання у всіх супутніх частинах коду.

Мікросервісна архітектура – це підхід до створення додатків, що має на увазі відмову від єдиної, монолітної структури. Тобто, замість того щоб виконувати всі обмежені контексти додатки на сервері за допомогою внутрішньо-процесних взаємодій, ми використовуємо кілька невеликих додатків, кожен з яких відповідає якомусь обмеженому контексту [5]. Причому ці програми працюють на різних серверах і взаємодіють один з одним по мережі, наприклад за допомогою HTTP або AMQP.

Основні переваги мікро-сервісної архітектури:

- *Гнучкість та ефективність роботи*. Розробники мікросервісу повністю розуміють його кодову базу. Вони можуть створювати, розгортати та тестувати його в незалежності від інших компонентів, використовуючи більш короткі ітераційні цикли. Мікросервіс – це лише один із багатьох компонентів системи, тому для його розробки можна використовувати найбільш відповідний технологічний стек для необхідної функціональності, а також оптимальних механізм стійкості. Даний підхід дозволяє суттєво знизити об'єм коду і істотно спрощує його підтримку. Це дозволяє окремим командам розробників швидко застосовувати нові технології чи модифікації існуючих технологій, не очікуючи поки інші фрагменти системи будуть готові до такого переходу [6]. Деякі визначення підкреслюють, що мікросервіс має бути настільки простим, щоби його можна було переписати за одну ітерацію.

- *Масштабованість*. Команда розробників мікросервісу може масштабувати компонент під час виконання в незалежності від інших мікросервісів, тим самим забезпечуючи ефективне використання ресурсів і швидке реагування на зміни навантаження. Теоретично навантаження компонента можна перенести на найбільш підходящу для даної задачі платформу. Завдяки такому незалежному переносу можна отримати гарну перевагу за рахунок правильного розміщення у мережі. Мікросервіси з якісним вихідним кодом надають величезні можливості для масштабування на вимогу, що успішно підтверджується практикою. Крім того, такі мікросервіси здатні реалізувати переваги гнучких хмарних середовищ, що забезпечують економічний доступ до величезного об'єму ресурсів.
- *Стійкість*. Окремі середовища виконання забезпечують стійкість сервісу, що не залежить від збоїв інших компонентів системи. При правильному розділі на компоненти кожен мікросервіс можна розроблювати під певні вимоги доступності, не зачіпаючи систему у цілому. Різні технології такі як, контейнери та спрощені середовища виконання дозволяють швидко відключати мікросервіси у випадку збоїв, без збитку для інших непов'язаних функцій. В той же час у реалізаціях мікросервісів не використовується модель зберігання поточного стану, що дозволяє миттєво перерозподілити навантаження і майже негайно ініціювати нові середовища виконання.

Серед інших переваг мікросервісної архітектури можна відзначити:

- *Незалежний технічний стек*: завдяки розподілу модулів за різними серверними вузлами і незалежної мови взаємодії ми можемо використовувати абсолютно різні мови програмування, інструменти взаємодії, моніторингу та зберігання даних. Це

дозволяє вибирати кращі і найбільш зручні рішення, а також експериментувати з новими технологіями.

- *Незалежна еволюція підсистем*. Мікросервіс може розвиватися і ламати зворотну сумісність, не обтяжуючи себе підтримкою старих версій, так як завжди можна залишити стару версію мікросервісу працювати протягом необхідного часу.
- *Збереження модульності.* І єдина, і мікросервісна архітектури дозволяють зберігати модульність і інкапсуляцію. Однак це може бути досить важким завданням, на вирішення якого підуть десятиліття, незважаючи на правила SOLID. Зате міякросервіси дозволяють забезпечувати логічний поділ програми на модулі за рахунок явного фізичного поділу по серверам. Фізична ізольованість захищає від порушення меж обмежених контекстів.

Недоліки:

- Стеко-незалежність може привести до хаосу всередині проекту.
- Необхідно забезпечувати зворотню сумісність системи.
- Необхідність знання оркестрації мікросервісів.
- Складність організації транзакцій.
- Відмово-стійкість до мережевих помилок.

Архітектура системи на основі мікросервісів вимагає іншого підходу до проектування. Для досягнення найкращих результатів необхідно наступне:

- Врахувати можливі побічні ефекти асинхронного зв’язку, якщо було прийняте рішення про відмову від залежних компонентів.
- Переконатися, що логіка системи не залежить від зберігання стану – це необхідно для того, щоб реалізувати відчутні переваги швидкого масштабування.
- Врахувати затримки у мережі при послідовній взаємодії.

- Врахувати обмежень на обробку помилок при використанні HTTP/JSON у відмінності від внутрішньої взаємодії.

Проаналізувавши переваги та недоліки кожної з вищенаведених архітектур систем та враховуючи вимоги пов’язані з масштабуванням, гнучкістю та стійкістю системи було обрано мікросервісну архітектуру для реалізації платформи.

2.2. Порівняння платформи Node.js та мови програмування Python

Node.js – це програмна платформа, з відкритим вихідним кодом, заснована на рушії V8, що транслює команди мовою JavaScript у машинний код. Завдяки Node.js стало можливим використання мови JavaScript на сервері, що перетворило її з вузько-спеціалізованої мови програмування для користувацьких інтерфейсів у мову загального призначення. Вона була свого роду квантовим стрибком, оскільки це перше середовище, що дозволяє використовувати JavaScript як на стороні клієнта, так і на стороні сервера. Без Node.js, клієнтська та серверна частина використовують різні мови, які можуть викликати певні проблеми сумісності [7].

Так як Node.js побудована на середовищі виконання JavaScript Chrome, вона допомагає розробляти масштабовані мережеві програми. Вона використовує керовану подіями неблокувальну модель вводу-виводу, що робить її ідеальним варіантом для розробки додатків реального часу з великою кількістю даних. Node.js пропонує більшу продуктивність і швидкість. Це ідеальне рішення для розробки програм обміну повідомленнями або спілкування в чаті. Це також корисно для розробки важких додатків і сайтів електронної комерції, які залежать від швидкості обробки [8]. Важливою деталлю є той факт, що дана платформа має дуже активну спільноту та неймовірно велику екосистему, завдяки чому є дуже легкою у розробці так як містить велику кількість допоміжних пакетів та

бібліотек які можна легко підключити та використовувати у своєму проекті [9].

Python – це динамічна, об'єктно-орієнтована мова програмування, яку порівнюють з подібними мовами Microsoft .NET або Java, як субстрат загального призначення для декількох видів розробки програмного забезпечення [10]. Вона забезпечує сильну підтримку інтеграції з кількома технологіями та більш високу продуктивність програмування протягом всього життєвого циклу розробки. Python особливо підходить для великих і складних проектів зі зміною вимог. У Python є бібліотека інструментів для аналізу даних з відкритим вихідним кодом, веб-фреймворки та інструменти тестування. Таким чином вона має одну з найбільших екосистем у порівнянні з будь-якими програмними спільнотами. Python також відрізняється послідовністю, стабільністю і легкістю використання. Це є оптимальним варіантом для випадку розробки наукових додатків, рішень для обробки великих даних та урядових програмних комплексів.

Проведемо порівняння Python та Node.js за ключовими показниками, що є важливими для розроблюваної системи.

Продуктивність та швидкодія. Головне, що вимагається від інструменту програмування – це продуктивність. Продуктивність визначає швидкість відповіді програми, яка безпосередньо залежить від швидкості виконання коду. Якщо порівняти продуктивність Python та Node.js, відразу видно, що Node.js є значно швидшою. Ця велика продуктивність значною мірою пояснюється тим, що Node.js базується на Chrome V8, який є дуже швидким і потужним рушієм. З цієї причини швидкість Node.js проти Python набагато більша, що робить його набагато кращим вибором для додатків у реальному часі. Крім того, Python не працює оптимально в додатках, що вимагають великої кількості пам'яті.

Масштабованість і паралельність. Іншим важливим аспектом, який необхідно враховувати при плануванні розробки серверної частини, є

масштабованість. Масштабованість – це здатність програми задовольняти зростаючу кількість запитів без компромісу в продуктивності. Ця здатність має важливе значення у важких додатках, а також у тих, які обслуговують декількох нових користувачів, як через настільні, так і на мобільні інтерфейси.

Мова програмування також впливає на масштабованість програми. Node.js створює однопоточну асинхронну архітектуру з операціями введення/ виводу, завершеними поза потоком [11], а, отже, не блокуючи її. Ця функція гарантує плавну масштабованість Node.js у простих веб-додатках, однак розробка складних додатків з безліччю паралельних процесів вимагає глибоких знань, уваги та ретельного інженерного дослідження.

Python, у свою чергу, не підтримує асинхронне програмування за замовчуванням, але підтримує перетворення, з якими легко можна домогтися асинхронної обробки. Отже, навіть якщо його архітектура може здатися не настільки масштабованою, як у Node.js, Python має інструменти, з якими можна досягти необхідної масштабованості [12].

Обробка помилок. Легкість і прозорість обробки помилок також може бути критичним питанням при виборі мови програмування. Як Node.js, так і Python добре справляються з генерацією та обробкою виняткових ситуацій, тобто помилок, що виникають під час виконання коду. Хоча, загальне поняття полягає в тому, що обробка винятків у Python простіше і займає менше часу для налагодження та виправлення помилок.

Як видно з вищенаведеного аналізу та порівняння Python та Node.js обидва мають свої переваги та недоліки, які визначають сфери їх застосування.

Вибір основної технології для проекту залежить від декількох факторів: типу програми, галузевого сектора, вимог до продуктивності,

досвіду розробників тощо. І Node.js, і Python забезпечать надійні, послідовні та ефективні рішення з правильним підходом.

Враховуючи тип системи, а також необхідність її швидкодії, стабільності, а також масштабованості було обрано платформу Node.js для написання її серверної частини.

2.3. Аналіз та порівняння СКБД

Система керування базами даних – сукупність програмних і лінгвістичних засобів загального або спеціального призначення, що забезпечують управління створенням і використанням баз даних. Система забезпечує безпеку, надійність зберігання і цілісність даних [13]. СКБД складається з таких основних компонентів:

- ядро, що відповідає за керування даними у зовнішній і оперативній пам’яті та журналізацію;
- мовний процесор бази даних, що забезпечує оптимізацію запитів на вилучення та зміну даних і створення, як правило, машинно-незалежного виконуваного внутрішнього коду;
- підсистему підтримки часу виконання, яка інтерпретує програми маніпуляції даними, що створюють користувальницький інтерфейс із СКБД;
- сервісні програми (зовнішні утиліти), що забезпечують ряд додаткових можливостей по обслуговуванню інформаційної системи.

Система має містити документо-орієнтовану СКДБ для зберігання основних сутностей системи та зручного управління ними. Оскільки система побудована на мікросервісах, необхідно налагодити механізм спілкуванням між ними. Після аналізу різних механізмів спілкування між мікросервісами та врахувавши інші вимоги для системи пов’язані з кешуванням даних та великою кількістю записів та зчитувань за ключем

стало очевидним та доцільним використання резидентної бази даних, за допомогою якої можуть бути реалізовані вище вказані вимоги. Оскільки система має підтримувати інтерактивну аналітичну обробку даних, то вона має містити також СКБД з підтримкою OLAP.

Проаналізувавши вимоги до роботи з даними та особливості їх обробки та використання у системі було обрано три типи СКБД для оптимального покриття вимог до роботи з даними.

2.3.1. *Порівняння аналітичних СКБД*

Аналітичні СКБД призначені для зберігання і обробки великих обсягів даних: від одиниць до сотень терабайт даних. Такі СКБД найчастіше використовуються для предиктивної аналітики, регулярної звітності, аналізу відтоку клієнтів, побудови корпоративних сховищ даних [14].

Одна з найважливіших якостей аналітичної СКБД – гнучкість і продуктивність при обміні даними з зовнішніми системами. Зокрема, в таких системах реалізовано протокол, який забезпечує взаємодію із зовнішньою системою одночасно всіх сегментів кластера. Якщо система-джерело також є кластер, то можна використовувати кластерну взаємодію з обох сторін, що дозволяє підвищити продуктивність, причому швидкість взаємодії буде зростати в міру розширення кластерів. Гнучка система резервування дозволяє розгорнути кластер з наперед заданим рівнем відмовостійкості, дозволяючи СКБД працювати навіть при виході з ладу половини серверів з кластера. Для порівняння далі буде розглянуто дві аналітичні СКБД – Vertica та Clickhouse.

Clickhouse – це стовпчикова аналітична СКБД з відкритим кодом, що дозволяє виконувати аналітичні запити у режимі реального часу на структурованих великих даних [15]. Clickhouse використовує свій особистий діалект SQL, що є дуже близьким до стандартного, але містить різні розширення такі як: масиви і вкладені структури даних, функції вищого

порядку, можливість роботи з зовнішніми key-value сховищами, спеціалізовані агрегатні функції, функції для приблизних обчислень та створення представлень з агрегацією, що зберігаються [16].

Переваги:

- можливість роботи з зовнішніми словниками;
- можливість зберігати результати агрегації для швидкого доступу до них;
- велика кількість рушіїв для різних типів задач;
- швидкодія.

Недоліки:

- відсутність транзакцій;
- обмежена підтримка синтаксису JOIN;
- відсутність точкових UPDATE/DELETE операцій;
- необхідність явного приведення типів.

Vertica – це стовпчикова аналітична СКБД, призначена для обробки великих обсягів даних, що забезпечує дуже швидку продуктивність запитів у традиційно інтенсивних сценаріях. Продукт покращує продуктивність запитів у порівнянні з традиційними системами реляційних баз даних, забезпечує високу доступність і масштабованість на петабайти даних. Vertica є відмовостійкою, в ній немає керуючої ноди: усі ноди рівні [17]. Якщо з одним з серверів в кластері виникають проблеми, то дані ми все одно отримаємо. Якщо розглядати Vertica на рівні кластера, то ця СКБД забезпечує масивно-паралельну обробку даних (MPP) в розподілено-обчислюваній архітектурі – де, в принципі, будь-який вузол готовий підхопити функції будь-якого іншого вузла [18].

Переваги:

- відмовостійкість кластеру;
- відсутність єдиної точки підключення;
- наявність проекцій;

- зріла екосистема з великою кількістю адаптерів та інтерфейсів, що підтримуються.

Недоліки:

- проблеми з масштабуванням;
- погана оптимізація проекцій.

2.3.2. *Порівняння резидентних СКБД*

Резидентна СКБД – система керування резидентними базами даних, тобто такими, що розміщенні у оперативній пам’яті [19].

Резидентні СКБД за рахунок оптимізацій, можливих в умовах зберігання і обробки в байто-адресованій оперативної пам'яті, забезпечують кращу швидкодію, ніж СКБД, що працюють з базами даних на пристроях постійного зберігання, як правило, з блочною організацією, і підключаються по шинним або мережевих інтерфейсах. При цьому розмір резидентної бази даних обмежений ємністю оперативної пам'яті вузла. Для ряду резидентних СКБД реалізуються техніки реплікації і сегментування, що дозволяють працювати з єдиною резидентної базою даних на декількох вузлах.

Оскільки оперативна пам'ять є енергозалежною, то використовується запис з попередньою журналізацією на енергонезалежному пристрої для забезпечення цілісності бази даних при раптовому перезавантаженні, тобто, робота з резидентної базою не виключає залежності від продуктивності підсистеми вводу-виводу (хоча і знижує її).

Найбільш поширеними та популярними резидентними СКБД є Redis та Aerospike, розглянемо їх більш детально та проведемо порівняння.

Redis – резидентна СКБД класу NoSQL з відкритим кодом, що працює зі структурами типу «ключ-значення». Використовується як для баз даних так і для реалізації брокерів повідомлень та кешування [20].

Переваги:

- має відкритий код та активну спільноту;

- легко встановлюється та не має залежностей;
- підтримка великої кількості різних типів даних.

Недоліки:

- відсутність мови запитів;
- данні мають поміщатись до оперативної пам’яті;
- необхідність вивчати Lua у випадку використання збережених процедур.

Aerospike – резидентна СКБД класу NoSQL, є сховищем типу «ключ-значення» з моделлю даних, що не містить схеми. Данні розділені за контейнерами доступу, просторами імен, які є семантично схожими з базами даних у реляційній СКБД. У просторі імен дані розподілені за множинами (аналог таблицям у реляційній СКБД) та записами (аналог рядкам у таблиці). Кожен запис має індексований ключ, унікальний у наборі, і один або кілька іменних стовпців, які утримують значення, пов'язані з записом [21].

Переваги:

- просте налаштування кластеризації;
- підтримка складених типів даних;
- агрегаційна модель запитів;
- можливість задати час, протягом якого дані будуть збережені у системі, після чого зникнуть;
- швидке перезавантаження системи.

Недоліки:

- безкоштовна версія має обмежену функціональність;
- усі ключі мають бути збережені у оперативній пам’яті не дивлячись на те де саме збережений простір імен.

2.3.3. *Порівняння документо-орієнтованих СКБД*

MongoDB – документована СКБД з відкритим кодом, що не потребує опису таблиць. Використовує формат BSON для зберігання документів,

який розширює реалізацію JSON для надання додаткових типів даних, таких як бінарні масиви та Date(для роботи з датою), а також забезпечує перевірку даних на основі стандарту JSON [22].

MongoDB є схематичною конструкцією, тобто кожен документ може мати свій власний набір унікальних полів в межах однієї колекції. Крім того, MongoDB поширюється і легко масштабується горизонтально для кращої продуктивності.

Переваги:

- не має схем (не вимагає уніфікованої структури даних);
- поєднує у собі переваги реляційних та документованих баз даних, а також баз типу «ключ-значення»;
- легка у налаштуванні реплікація та масштабування [23].

Недоліки:

- слабкий повнотекстовий пошук;
- обмеження розміру документу у 16Мб.

CouchDB – документо-орієнтована система управління базами даних з відкритим вихідним кодом, яка не потребує опису схеми даних, поширюється вільно, написана на мові Erlang. Для зберігання даних використовується JSON, а для реалізації MapReduce-запитів – JavaScript. Можливе написання будь-якої логіки на Erlang [24]. Однією з особливостей СКБД є підтримка реплікації з декількома провідними вузлами. CouchDB можна розглядати як сервер веб-додатків, для реалізації цієї ідеї в CouchDB вбудований продуктивний веб-сервер, а програмний код, як і дані, зберігається в тій же базі даних. Для автоматизації роботи з додатками використовується утиліта CouchApp.

Дотримуючись підходу NoSQL, CouchDB не зберігає дані і зв'язки в таблицях. Замість цього кожна база даних – набір незалежних документів. Кожен документ містить свої власні дані і незалежну схему. Додаток може отримати доступ до кількох баз даних, наприклад, що зберігаються на

мобільному телефоні користувача і на сервері. Метадані документа містять інформацію про версії, дозволяючи об'єднувати дані і вирішувати будь-які протиріччя, які могли з'явитися в момент, коли бази даних були роз'єднані [25].

Для управління конкурентним доступом використовується механізм MVCC, завдяки чому можливо уникнути необхідності блокування файлу бази даних під час запису. Вирішення конфліктів відноситься до сфери відповідальності логіки додатка, вирішення конфлікту зазвичай включає в себе об'єднання даних в один документ, а потім старий документ видаляється.

Переваги:

- легкість налаштування реплікацій;
- підписка на зміни даних.

Недоліки:

- не індексовані запити є ресурсо-затратними;
- потребує більше місця для зберігання даних, у порівняні з аналогами.

2.4. Аналіз рекламних серверів

Рекламний сервер – це рекламна технологія, яка використовується видавцями, рекламодавцями, рекламними агентствами та рекламними мережами для керування та проведення рекламних кампаній у мережі Інтернет. Рекламні сервери несуть відповідальність за прийняття миттєвих рішень про те, які оголошення відображатимуться на веб-сайті, а потім подають їх. Крім того, сервер оголошень збирає та повідомляє дані (наприклад, покази, кліки тощо) для рекламодавців, щоб отримати інформацію про продукти та переглянути ефективність своїх оголошень.

Рекламні сервери дозволяють рекламодавцям:

- зберігати та керувати кодами оголошень;

- налаштовувати критерії відстеження для рекламних кампаній;
- відстежувати ефективність (покази, кліки, конверсії тощо) всієї кампанії в усіх видавцях в єдиній системі;
- вимірювати охоплення кампанії, беручи до уваги спільну аудиторію видавців;
- перевіряти звіти, надані видавцями;
- допомагають оптимізувати кампанії.

Всі ці функції є невід'ємною частиною ведення рекламної кампанії в Інтернеті, щоб переконатися, що рекламний контент відображається там, де і для кого він призначений. Це також допомагає при аналізі побачити, наскільки ефективною є кампанія і чи містить контент та має бажані результати. Рекламні сервери також пропонують більш розширені функції для більш складних рекламних кампаній. Додаткові функції включають обмеження частоти показів, визначення послідовності оголошень (також називаються сеансами оточення), оптимізацію пошукової системи та націлювання. Рекламодавці також можуть обмежувати об’яви, встановлюючи обмеження на витрати грошей.

Сьогодні на ринку рекламних серверів представлена велика кількість гравців у яких однією основною відмінністю є аудиторія. Серед них перше місце розділяють рекламні сервіси Google, YouTube та Facebook які мають найбільший обсяг аудиторії та досконалу систему таргетування, що робить їх ідеальним варіантом для поширення своєї реклами.

Google DCM (Google Campaign Manager) – це рішення для показу реклами та управління нею, що дозволяє агенствам та рекламодавцям контролювати усі цифрові рекламні програми [26]. Campaign Manager спрощує планування кампаній, розміщення, таргетинг, показ та оптимізацію реклами, а також створення звітів.

Facebook Ads Manager – це інструмент, за допомогою якого можна створювати рекламу та керувати нею у платформі Facebook [27]. За

допомогою нього можна створювати та показувати рекламу, налаштовувати таргетинг на потрібну аудиторію, налаштовувати бюджет, відслітковувати результативність реклами та кампанії у цілому, переглядати історію платежів, інформацію про способи оплати та зібрану статистику.

YouTube Ads – інструмент для створення та керування рекламою у платформі YouTube. Також надає можливості для створення та управління рекламними кампаніями та рекламним інвентарем [28].

Усі три інструменти мають дуже схожу функціональність, але кожен з них працює зі своєю платформою.

3. СТРУКТУРНО-АЛГОРИТМІЧНА ОРГАНІЗАЦІЯ

3.1. Загальна структура програмних засобів

Сервіс API є головним сервісом системи та виконує наступні функції:

1. Авторизація та контроль доступу до системи.
2. Видача користувацького інтерфейсу.
3. Направлення задач на мікросервіси та їх контроль.
4. CRUD для більшості сутностей системи.
5. Агрегація даних з різних джерел та забезпечення прикладного інтерфейсу для доступу до них.

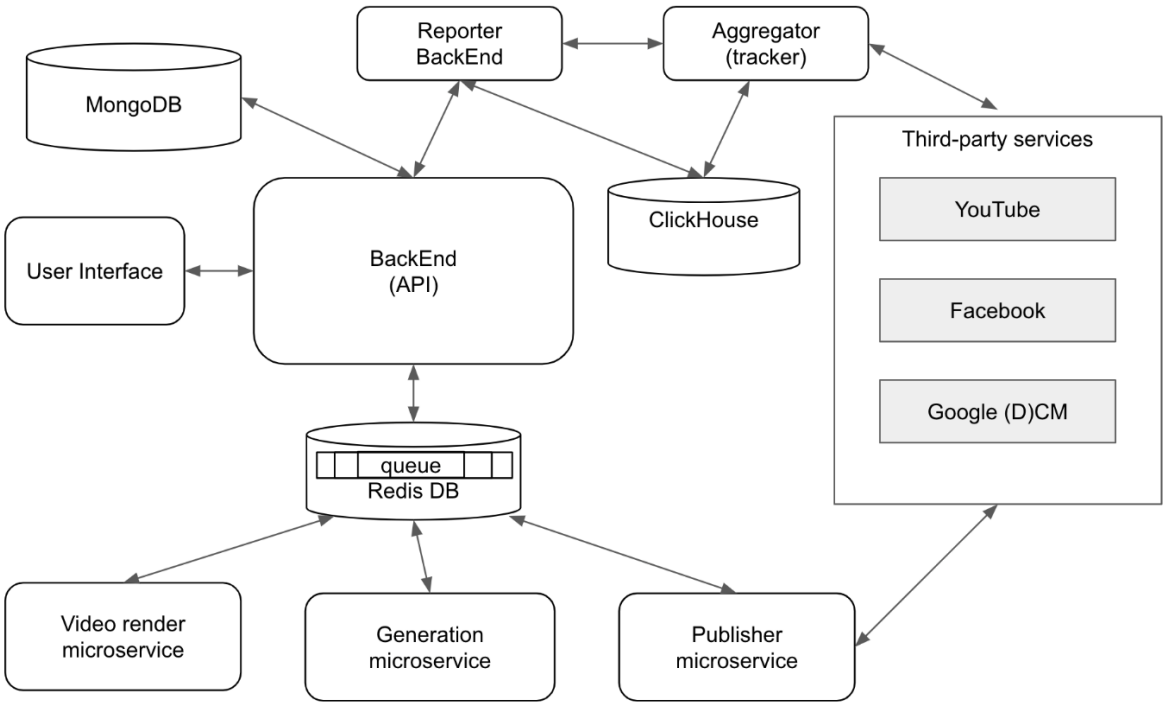


Рис. 3.1. Загальна структура системи

Клієнтський інтерфейс взаємодіє з користувачем, формує та посилає запити до сервера, а також показує дані, що були отримані від сервера. Головний сервіс взаємодіє з мікросервісами через повідомлення, що

передаються їм чергою. Кожен мікросервіс знає структуру отриманого повідомлення та як його інтерпретувати.

Мікросервіс генерації варіацій отримає команду від сервісу API коли користувач створює дані для варіацій. Він генерує варіації необхідного формату та зберігає їх у базі даних.

Мікросервіс публікації креативів також отримує команду на публікацію від сервісу API. Його задачею є отримання готових до публікації варіацій їх попередня обробка, пристосування до формату необхідного каналу для публікації та їх завантаження.

Мікросервіс трекінгу та агрегації даних, отримує на вхід події з показуваних креативів, а також статистику з каналів для публікації, агрегує їх та записує до бази даних ClickHouse.

Загальну структуру системи зображено на рис. 3.1.

3.2. Структура бази даних

Проаналізувавши дані, з якими має працювати система, а також сценарії та частоту доступу до цих даних було виявлено необхідність використання трьох СКБД різних типів:

- об’єктно-документної;
- аналітичної;
- резидентної.

Як об’єктно-документну СКБД було обрано MongoDB. Вона відповідає за збереження основних сутностей системи та взаємозв’язок між ними.

Як аналітичну СКБД для зберігання великої кількості даних статистики та агрегування їх за різними параметрами було обрано Clickhouse

Як резидентну СКБД було обрано Redis за допомогою якої буде налаштоване кешування та черга повідомлень.

3.2.1. *Структура MongoDB*

Колекція users містить основну інформацію про користувачів, що є необхідною для їх авторизації у системі та розподілення по ролях. Містить такі поля:

- логін користувача;
- пароль користувача;
- набір ролей, що присвоєні даному користувачеві.

Для реалізації міграцій у системі є колекція migrations, документи якої містять ідентифікатор сценарію міграції та час коли він виконався. Використовується для того, щоб знати поточний стан міграцій та знати з якої міграції продовжити виконання. Колекція advertisers складається з документів, що містять інформацію необхідну для інтеграції з рекламними серверами:

- назва рекламодавця;
- ідентифікатор профілю користувача у системі Google DCM;
- ідентифікатор рекламодавця користувача у системі Google DCM;
- ідентифікатор рекламного акаунту користувача у системі Facebook Ad Manager.

Колекція audiences (рис. 3.2) містить інформацію про таргетинги, що були задані оператором для набору варіацій, має наступні атрибути:

- назва аудиторії;
- ідентифікатор рекламної кампанії, якій належить дана аудиторія;
- список таргетингів для каналу Google;
- список таргетингів для каналу Facebook.

Правила для таргетингів зберігаються у вигляді пар «ключ-значення», та не мають чіткої структури, адже для кожного з каналів для публікації вони є різними.

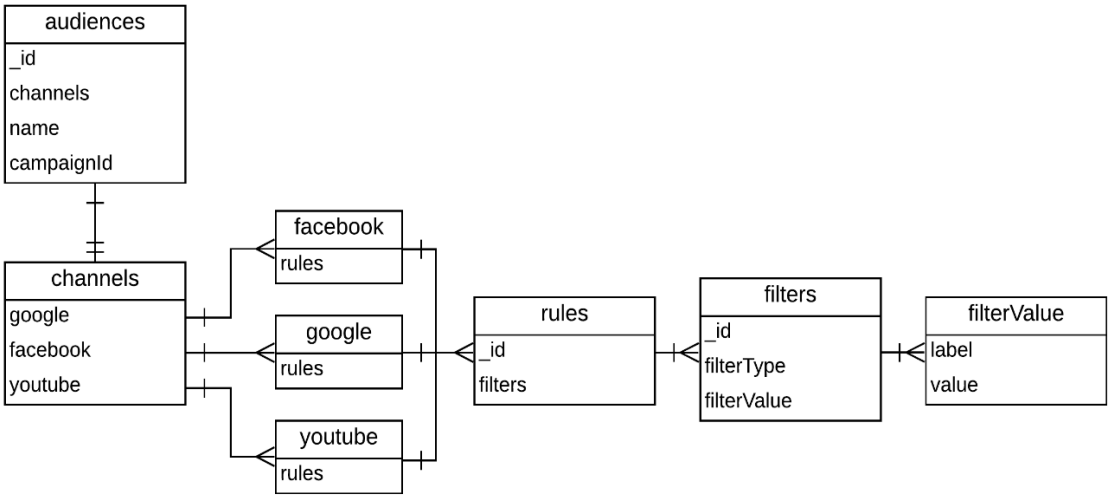


Рис. 3.2. Структура колекції audiences

Колекція briefs (рис. 3.3) містить інформацію про побажання клієнта щодо того, якою має бути створена кампанія. Включає в себе наступу інформацію:

1) назва майбутньої кампанії;
2) бажана дата початку;
3) обрані формати для креативів;
4) контакти учасників:
    - контакти команди;
    - контакти партнерів;
    - контакти медіа-команди;
5) тип метрики для кампанії;
6) активи для кампанії:
    - завантажені файли;
    - посилання на файли;
    - типи активів (відео, аудіо, шрифти і т.д.).

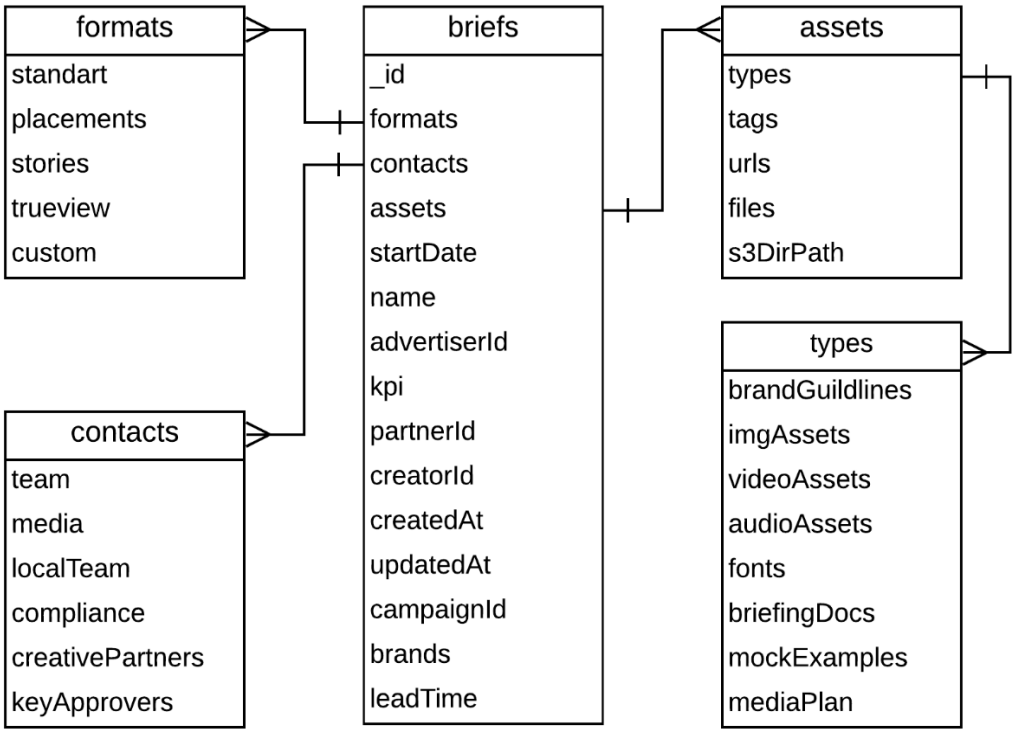


Рис. 3.3. Структура колекції briefs

Дані про створенні рекламні кампанії зберігаються у колекції campaigns документи якої містять наступну інформацію:

- назва;
- ідентифікатор бріфу на основі якого було створено кампанію;
- дати старту та завершення кампанії;
- обрані канали поширення рекламних креативів;
- шаблони, джерела даних та активи для креативів;
- ідентифікатори створених кампаній у системах Google DCM та Facebook Ad Manager;
- конвенція імен для реклами та розміщень у Google DCM.

3.2.2. *Структура Redis DB*

Ця база даних використовується як кеш-сховище, а також як черга повідомлень, що має свій протокол та структуру всередині. Кеш-сховище

працює як набір пар «ключ-значення», де за певним ключем у кеші знаходиться необхідне значення.

3.3. Опис основних мікросервісів системи

До мікросервісів винесено задачі системи які займають багато ресурсів та часу, що потребується для завершення виконання. До таких задач належать рендеринг відео на основі проекту After Effects, генерація варіацій даних для креативів, публікація креативів до різних рекламних майданчиків та інше. Такі задачі поділено по групам за виконуваною функцією та часом виконання, для того, щоб максимально доречно використовувати мікросервіси та досягнути мінімальної затримки.

3.3.1. *Мікросервіс генерації варіацій*

Зоною відповідальності цього мікросервісу є автоматизація створення даних для рекламних креативів. Даний мікросервіс генерує набори даних для майбутніх рекламних креативів на основі завантаженого шаблону, що містить статичні дані, а також динамічні поля, яким будуть присвоєї різноманітні значення на основі підвантаженних активів.

Шаблон, активи з даними та правила для генерації варіацій задаються адміністративним користувачем у системі. Для кожного динамічного поля, в залежності від його типу, можуть бути завантажені певні дані. Для текстового поля може бути задане деяке слово чи файл зі списком слів. А для динамічних медіа полей – зображення у певному форматі. Адміністратор може додавати нові шари у вигляді рядків та задавати правила використання певного поля чи їх набору. Вони можуть бути у заблокованому чи розблокованому стані, в залежності від якого змінюється їх використання у різних комбінаціях. Наприклад, якщо заблокувати декілька полів у одному шарі, то вони будуть поширені тільки разом у одній

комбінації, а у випадку розблокованих полів, вони будуть поширюватися у всіх можливих для них комбінаціях.

Мікросервіс працює у фоновому режимі та чекає на команду від сервісу API, щоби почати генерувати дані. Мікросервіс генерації та API спілкуються та передають команди за допомогою черги повідомлень.

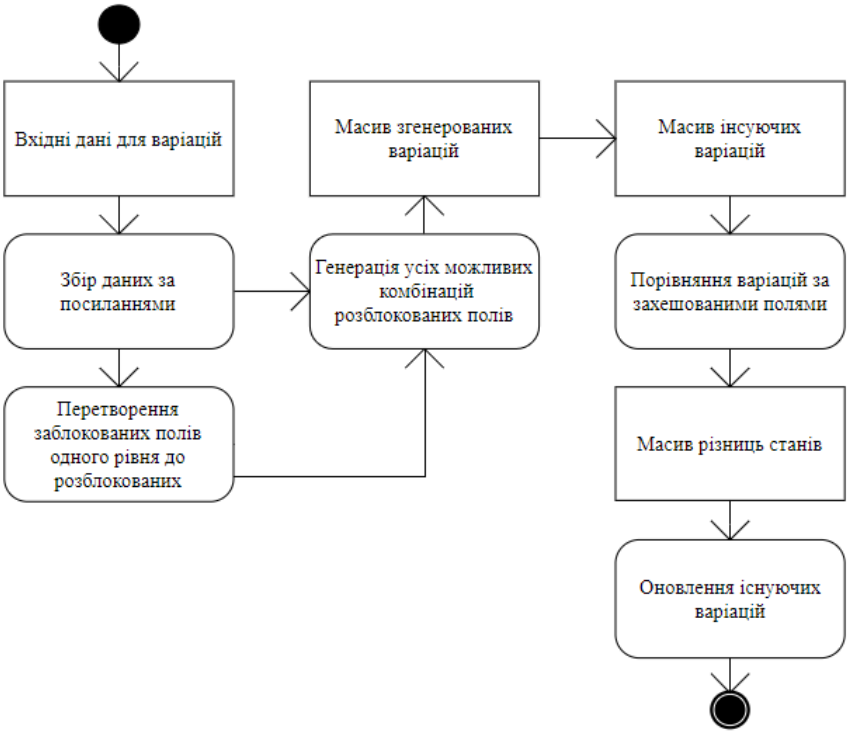


Рис. 3.4. Схема алгоритму генерації варіанцій

Коли створено новий набір даних для варіацій або модифіковано існуючий, API сповіщає мікросервіс генерації про те, що для кампанії з вказаним ідентифікатором були змінені вхідні дані і потрібно провести регенерацію варіацій. Для цього, одразу після зміни вихідної сутності з даними, API відправить повідомлення, що міститиме усю необхідну інформацію, таку як унікальний ідентифікатор кампанії та інші метадані.

Мікросервіс генерації варіацій, на основі ідентифікатора кампанії, отримає з бази даних необхідні вхідні дані або посилання на них для генерації варіацій, а також метадані, що відповідають за спосіб та правила

їх генерації. Першим кроком проводиться остаточний збір даних на основі існуючої у базі даних інформації та посиланням на інші дані. Далі для кожного з полів буде проаналізовано спосіб їх поширення, усі дані з заблокованим способом поширення помічаються для їх спільного подальшого використання. З отриманої агрегованої структури відбувається поширення варіацій у всіх можливих комбінаціях, що були задані правилами. Для кожної зі згенерованих варіацій вираховується унікальний хеш на основі даних, що входять до її складу. Наступним кроком мікросервіс отримує список раніше згенерованих варіацій, що належать кампанії із заданим ідентифікатором, та порівнює їх хеші з хешами щойно згенерованих варіацій. Це робиться для того, щоби система розуміла які варіації лишилися незмінними, а які з них треба додати до системи. Схема алгоритму зображено на рис. 3.4.

3.3.2. Мікросервіс публікації креативів

Даний мікросервіс отримує на вхід ідентифікатор кампанії, для якої треба виконати публікацію, а також канал, у який потрібно опублікувати обрані креативи. Для публікації реклами доступно три канали: Google, Facebook та YouTube, а також їх різні підвиди та комбінації, які можуть відрізнятись ціною, способом взаємодії та часом оновлення.

Публікація через платформу Google.

Мікросервіс підтримує три варіанти публікації до каналу Google:

1. Завантаження шаблону креативу до Google Ad Manager та файлу з вмістом креативів та таргетингами до Google Spreadsheets.
2. Завантаження шаблону креативу, створення сутностей для розміщення реклами та таргетингу у Google Campaign Manager (DCM) та завантаження файлу з вмістом креативів та ідентифікаторами створених сутностей.

3. Завантаження шаблону креативу, створення сутностей для розміщення реклами та таргетингу, генерація креативів у мікросервісі та їх публікація до Google Campaign Manager (DCM).

Публікація через Google Spreadsheet

У випадку якщо було обрано канал публікації Google та підвид публікації «Non Trafficked», то обрані креативи створюються за допомогою сервісу Google Spreadsheet. Спочатку до сервісу Google буде завантажено шаблон для креативу з динамічними полями. Далі для кожного креативу, що заснований на цьому шаблоні, система зчитає значення, що відповідають динамічним полям у креативі та заносить їх до csv файлу. Після того як файл сформовано і він містить інформацію про усі необхідні креативи він завантажується у спеціальну папку на Google Drive. Google Ad Manager раз на дві години перевіряє задану папку на Google Drive на наявність необхідного файлу та на його основі будує необхідні креативи.

Публікація через Google DCM

У випадку якщо було обрано канал публікації Google та підвид публікації «Trafficked» чи «Prerender», то обрані креативи публікуються за допомогою сервісу Google DCM. У разі вибору даних підвидів для публікації, адміністратор має задати конвенцію імен для реклами та розміщень – сутностями, якими оперує сервіс Google DCM. Реклама у контексті Google DCM відповідає за різноманітні таргетинги, тобто кому буде показаний рекламний креатив, а розміщення – за те де і як він буде показаним. Конвенція імен може включати статичний текст, а також макроси. На основі ідентифікатору кампанії буде отримано варіації, що були підтверджені клієнтом, та є готовими для публікації, а також задані конвенції імен. Генерація імен для реклами та розміщень на основі конвенції. Імена конвенції окрім статичного тексту у назві можуть мати два різні типи макросів. Перший тип – це макроси що відповідають таргетингам DCM, а другий – макроси, на місце яких будуть підставлені довільні

значення з csv файлу. Для всіх згенерованих місць для розміщень та реклами обчислюється хеш для значень обов'язкових макросів. Всі можливі комбінації для розміщень створюються у системі Google DCM з відповідними параметрами і у результаті отримуються їх ідентифікатори. Далі для кожного елементу реклами шукаються усі розміщення з такими ж хешами після чого у системі Google DCM створюються необхідні реклами з прив'язаними за ідентифікатором розміщеннями.

У випадку підтипу «Trafficked» до сервісу завантажується шаблон для креативу з динамічними полями. Для кожного креативу, що заснований на цьому шаблоні система зчитує його значення, що відповідають динамічним полям у креативі, а також значення ідентифікаторів реклами та розміщення, до якого ця варіація належить і записує їх до csv файлу.

У випадку підтипу «Prerender» мікросервіс генерує креатив для кожної варіації на основі її даних, а також задає креативу значення відповідної реклами та розміщення. Далі кожен згенерований креатив завантажується до системи і є готовим до показу.

Публікація через платформу Facebook

Для платформи Facebook мікросервісом підтримується два види публікації «Facebook Direct» та «Facebook with CM Tracking».

У випадку «Facebook Direct» Мікросервіс отримує на вхід ідентифікатор кампанії, для якої необхідно виконати публікацію Facebook креативів певного виду. Мікросервісом публікації підтримуються наступні типи креативів для платформи Facebook: Single Image, Single Video, Dynamic Video. За ідентифікатором кампанії мікросервісом отримується список варіацій дозволених до публікації. Наступним кроком отримуються необхідні дані для варіацій та замінюються даними у форматі необхідними для публікації. На основі отриманих даних варіації розподіляються за існуючими рекламними групами. Для кожної із рекламних груп отримується список вже створеного рекламного інвентарю у платформі Facebook. Далі

порівнюються реклами, що увійшли до рекламної групи та ті що вже завантажені до Facebook. Кожна існуюча реклама, що була отримана з сервісу порівнюється за допомогою хешу з тією, що була згерерована у мікросервісі. У випадку, якщо не знайдено ідентичного хешу до існуючої реклами у Facebook серед локально згенерованих реклам, то це означає, що рекламу за даним ідентифікатором необхідно вимкнути у платформі Facebook. Якщо серед локально згенерованих реклам існує така, що є вимкненою у Facebook, то рекламу за таким ідентифікатором необхідно увімкнути. А якщо серед локально згенерованих реклами існує така, що не міститься у Facebook, то її необхідно додати до системи.

Facebook with CM Tracking. Основною відмінністю цього методу від попереднього є те, що при публікації креативів для відслідковування показників переходів за креативом використовується Google Campaign Manager.

Основні кроки:

1. Для кожної варіації створити місце для розміщення формату 1x1 піксель, а також рекламу типу «Tracking».
2. Пов’язати між собою місце для розміщення та рекламу.
3. Створити рекламний креатив типу Tracking та пов’язати його з рекламною кампанією.
4. Пов’язати даний креатив з усіма створеними рекламами, включити його до ротації та для кожної комбінації задати йому посилання, що задане у варіації.
5. Згенерувати теги для відслідковування переходів на основі створеної структури та завантажити їх.
6. Для кожної структури отримаємо унікальний тег, що буде відслідковувати переходи та перенаправляти на посилання задане під час створене креативу.

7. Для кожної варіації замінити посилання на веб-ресурс на тег, що відслідковує переходи та перенаправляє на той самий ресурс.
8. На основі отриманих варіацій виконати публікацію «Facebook Direct» як описано у попередньому алгоритмі.

3.4. Сервіс API та робота з чергою

Основним сервісом системи є сервіс API, який забезпечує прикладний інтерфейс для взаємодії з різними сутностями. Його зоною відповідальності є авторизація та контроль доступу до системи, обробка запитів від користувацького інтерфейсу та керування іншими мікросервісами за допомогою черги повідомлень.

3.4.1. Алгоритм JWT авторизації

JSON Web Token – це об'єкт JSON, який є визначеним відкритим стандартом RFC 7519. Він вважається одним із безпечних способів передачі інформації між двома учасниками. Для його створення необхідно визначити заголовок з загальною інформацією щодо токену, основні дані, що передаються, такі як ідентифікатор користувача, його роль і т.д., а також підписи [29].

Важливо зауважити, що JWT не приховує та не маскує дані автоматично. JWT використовується для перевірки того, що відправленні дані були дійсно відправлені авторизованим джерелом.

Кроки для JWT аутентифікації (рис. 3.5):

1. Користувач вводить свій логін та пароль у форму на клієнтській частині.
2. Дані з клієнту передаються на сервер аутентифікації.
3. Сервер створює JWT токен та відправляє його користувачеві.
4. Коли користувач робить запити до API він добавляє до них отриманий раніше JWT токен.

5. Під час запиту до API сервер перевіряє токен, що був переданий разом із запитом. Він визначає чи дійсно являється користувач тим за кого себе видає, а також чи це саме той токен, що був виданий сервером аутентифікації.

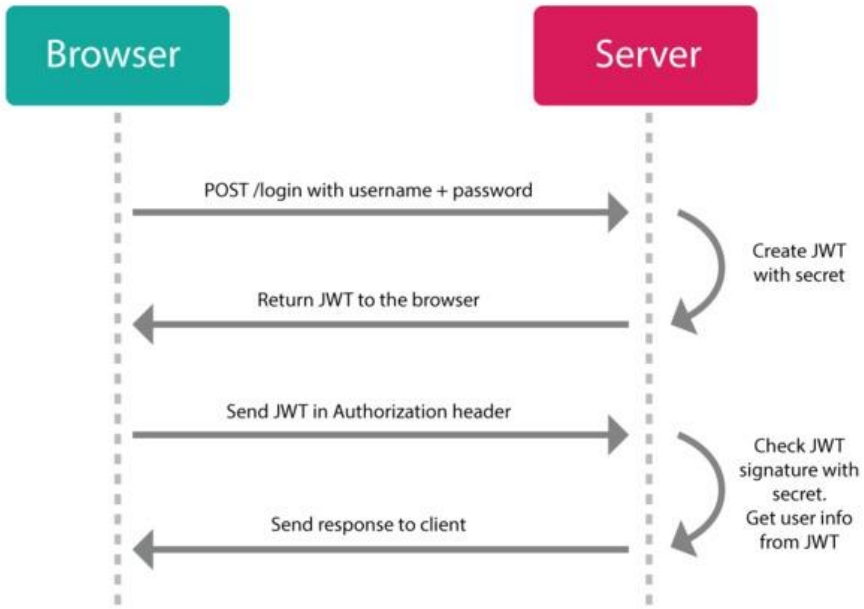


Рис. 3.5. Схема клієнт-серверної JWT авторизації

Перевірка JWT токена сервером

Секретний ключ зберігається на сервері та відомий тільки йому. Цей ключ використовується як під час видачі токена так і під час його перевірки при різноманітних запитах. Оскільки сервер знає секретний ключ, коли користувач робить API-запит з доданим до нього токеном, то він може виконати той же алгоритм підписування до JWT. Далі цей підпис буде порівняно з особистим підписом, що був підрахований алгоритмом хешування. У випадку якщо підписи співпадають, то це означає, що JWT токен є валідним, а інакше це символізує про те, що щось пішло не так і це може бути потенційною спробою атаки.

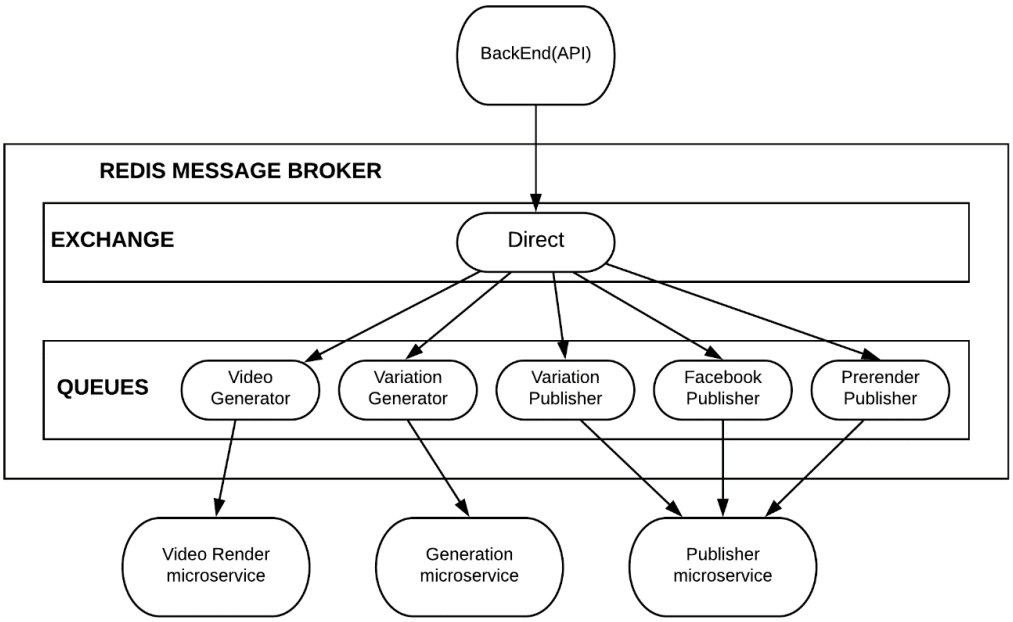


Рис. 3.6. Схема взаємодії мікросервісів

3.4.2. Міжсервісна взаємодія за допомогою Redis Message Broker

Черга повідомлень – програма, яка переводить повідомлення в офіційний протокол обміну повідомленнями між відправником та отримувачем [30].

Сервіс API взаємодіє з мікросервісами за допомогою брокеру повідомлень (рис. 3.6). Для кожної окремої задачі створена своя черга для того, щоб уникнути блокування та накопичення повідомлень у черзі. Наприклад, довелося створити окрему чергу повідомлень для prerender публікації, адже процес рендерингу креативів займає багато часу, у порівнянні з іншими задачами для публікації. Головний сервіс знає список існуючих черг, а також у яку чергу потрібно відправити повідомлення. Для відправлення повідомлення у конкретну чергу сервіс використовує ідентифікатор черги, а саме її ім’я. На основі цього ідентифікатору механізм розподілення повідомлень на черги визначає необхідну чергу та передає повідомлення до неї [31].

Повідомлення можуть мати різні статуси:

- Waiting: повідомлення чекає доки знайдеться обробник, що зможе його опрацювати;
- Active: виконується обробка даного повідомлення;
- Completed: повідомлення, що були успішно оброблені, такі повідомлення видаляються з черги після успішної обробки;
- Delayed: повідомлення виконання якого було відкладено;
- Failed: повідомлення у процесі обробки яких сталось помилка.

Кожен з мікросервісів працює у фоновому режимі, та чекає на повідомлення від головного сервісу. Спочатку повідомлення у черзі з’являється зі статусом Waiting, до тих пір поки не знайдеться вільний обробник для даного повідомлення. Коли повідомлення почне оброблюватися його статус змінюється на Active, далі воно буде знаходитися в такому статусі доки обробка не завершиться успішно і тоді повідомлення перейде до статусу Completed чи поки не станеться помилка у обробці та статус повідомлення буде змінений на Delayed або Failed. У випадку якщо під час обробки сталася не критична помилка, яка не обов’язково повториться під час наступного виконання, то статус повідомлення буде змінений на Delayed і мікросервіс спробує його опрацювати ще раз. Якщо сталася критична помилка і зрозуміло, що наступні спроби виконати таку задачу не закінчаться успіхом, то таке повідомлення буде переведено до статусу Failed і мікросервіс більше не намагатиметься його виконати ще раз.

4. АНАЛІЗ РОЗРОБЛЕНИХ ПРОГРАМНИХ ЗАСОБІВ

4.1. Загальна структура проекту

Оскільки система складається с багатьох мікросервісів та модулів, що є універсальними для всієї системи та використовуються у різних мікросервісах, то було вирішено зберігати усі існуючі модулі та мікросервіси у монорепозиторії. Монорепозиторій (монолітний репозиторій) – це стратегія розробки програмного забезпечення, де код для багатьох проектів чи сервісів одного проекту зберігається в одному сховищі. Основною складністю у випадку монорепозиторію є перевикористання спільних залежностей, а також управління усіма патеками системи. Для полегшення роботи з монорепозиторієм було використано бібліотеку Lerna – інструмент для керування JavaScript проектом, з великою кількістю пакетів.

Структура монорепозиторію:

```
packages/
    core/
    common/
    db/
    api/
    ui/
    publisher/
    generator/
    renderer/
package.json
lerna.json
```

Монорепозиторій містить файл lerna.json у якому вказане місце, де зберігаються пакети, у даному випадку це папка packages. У ній містяться папки з мікросервісами та універсальними модулями.

- core – модуль, у якому інкапсульована загальна логіка, що виконується у різних місцях системи;

- common – модуль, у якому зберігаються загальні константи, та допоміжні функції;
- db – модуль, у якому знаходяться моделі бази даних та міграції;
- api – основний сервіс системи;
- ui – модуль з клієнтським інтерфейсом;
- publisher – мікросервіс для публікації рекламних креативів;
- generator – мікросервіс для генерації варіацій;
- renderer – мікросервіс для генерації відео.

4.2. Опис дизайну та вмісту сторінок

Відповідно до вимог, дизайн сторінок виконано у стилі Material. Для написання клієнтської частини використовувалась бібліотека React, HTML5, JavaScript та препроцесор SCSS. Для реалізації усіх сторінок у стилі Material було використано бібліотеку MaterialUI. Усі сторінки є адаптивними з мінімально підтримуваним розширенням 768 пікселів.

У лівій частні сторінок розміщене меню (рис. 4.1), яке має набір закладок для сторінок доступних для даного типу користувача, а також конопку виходу з системи. У випадку головного адміністратора та адміністратора меню також містить випадаючий список акаунтів для партнерів. Меню можна переключити у мінімалістичний режим (рис. 4.1), де показуються лише іконки сторінок, щоб залишити більше місця для контенту.

Поточна сторінка у меню виділена рожевим кольором як для повного меню так і для мінімалістичного його вигляду.

Сторінка авторизації побудована з білої картки на тлі ніжно-лазурного кольору (рис. 4.2), що містить поля для вводу логіну і пароля, а також кнопку для підтвердження авторизації.

Сторінка для керування користувачами знаходиться на вкладці «Користувачі» для ролей Супер адміністратора та Адміністратора.

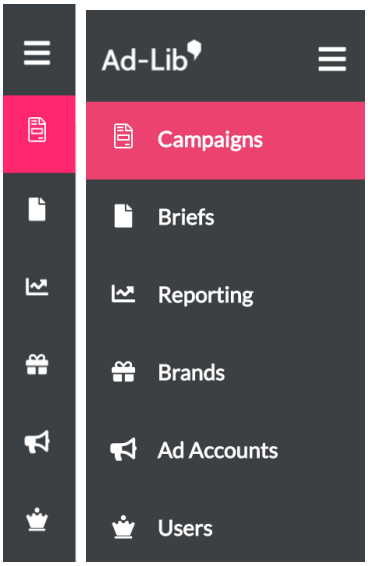


Рис. 4.1. Меню у повному та мінімалістичному вигляді

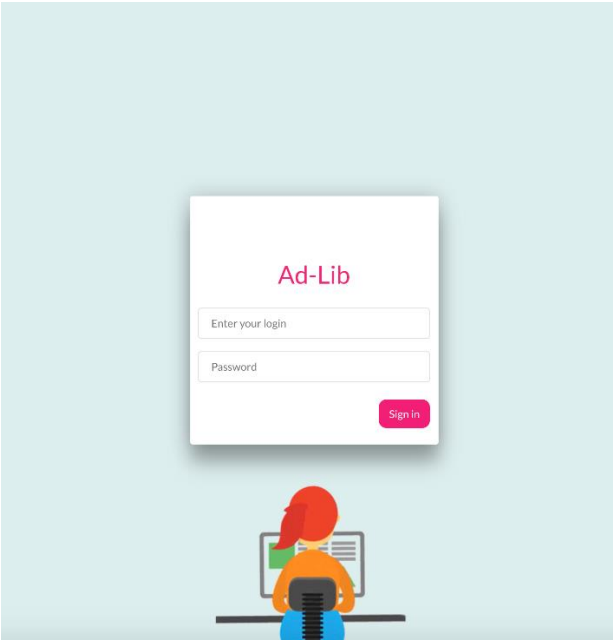


Рис. 4.2. Сторінка авторизації

Сторінка містить список існуючих користувачів у вигляді карток з інформацією про логін користувача та ролі, що йому присвоєні. При натисканні на картку відкриється сторінка редагування користувача, де в залежності від ролі користувача, що редагується, будуть доступні різні поля

для редагування. На сторінці зі списком користувачів у правому верхньому кутку є кнопка «створити користувача», що перенаправляє на сторінку створення користувача, де в залежності від обраної ролі у випадаючому списку буде завантажено різні форми для створення користувача.

На вкладці «Кампанії» розташовується сторінка для керуванням кампаніями (рис. 4.3), що є доступною тільки для користувачів з ролями Адміністратор та Головний Адміністратор. На цій сторінці міститься список усіх рекламних кампаній, що доступні даному користувачеві. Кожна кампанія представлена у вигляді картки з основною інформацією, такою як назва, дата початку та завершення, поточний статус (активна, завершена). Окрім цієї інформації картка з кампанією містить список каналів для публікації. Список представлений у вигляді логотипів, бо вони є впізнаваємими та займають мало місця. Для зручності та інтуїтивної зрозумілості логотипи активних каналів показані у кольорах, у той час як не активні – сірим кольором. У правому верхньому куті сторінки знаходиться кнопка «створити», що перенаправляє на сторінку створення нової рекламної кампанії. Користувачу пропонується задати такі основні дані як назва, дати початку та завершення, а також обрати зі списку рекламний акаунт, що є пов’язанним з нею. Також потрібно обрати один або декілька каналів для публікації рекламного інвентарю, для цього користувачеві доступні кнопки для додавання даного каналу до кампанії.

Після додавання каналу до кампанії, на сторінці з’являться нові поля для заповнення, що є необхідними для опису доданого каналу. Під формою знаходяться дві кнопки для підтвердження створення і відміни та повернення на список існуючих кампаній. У випадку спроби підтвердження без наявності всієї необхідної інформації користувач буде залишений на сторінці та йому будуть показані поля які є обов’язковими до заповнення. Після успішного створення користувач перенаправляє на список усіх кампаній, що включатиме щойно створену.

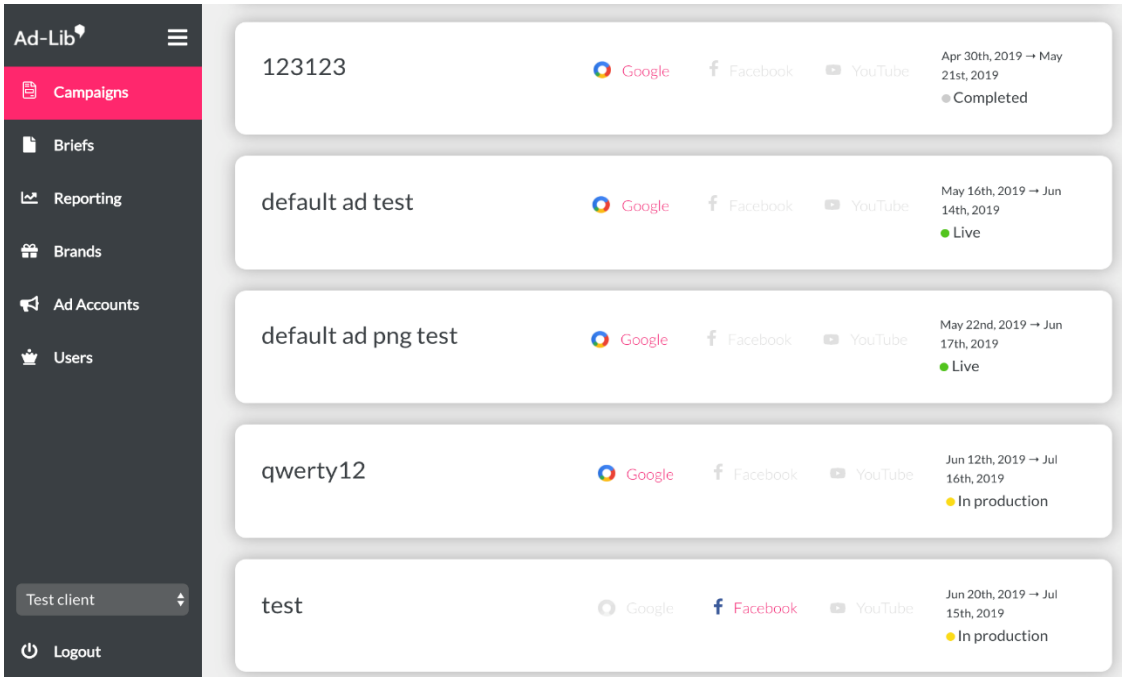


Рис. 4.3. Сторінка керування рекламними кампаніями

Якщо натиснути на картку однієї з таких кампаній, користувача буде перенаправленно на сторінку обраної кампанії. У верхній частині сторінки знаходиться назва поточної кампанії, а також заголовок з вкладками: Overview, Templates, Assets, Data Sources, Audiences, Assemblies, Variations, Approval. Усі вкладки зображені сірим кольором на білому заголовку. При наведені курсором на поле воно змінює свій колір на рожевий, а поточна вкладка виділена жирним шрифтом, та підкреслена рожевою лінією. На вкладці Overview знаходиться попередньо задана інформація про рекламну кампанію. Вкладка Templates містить поле для завантаження, а також уже завантажені шаблони для поточної кампанії у вигляді списку. Кожен такий елемент списку містить наступну інформацію: формат шаблону, приналежність до каналу поширення, а також список динамічних елементів, що були знайдені у цьому шаблоні. У правій частині знаходяться кнопки для видалення шаблону та його перегляду, що веде користувача на окрему сторінку. На ній користувач має можливість змінювати текстові динамічні

поля, та обирати доступні зображення, з якими вони хочуть переглянути шаблон і подивитись на результат. Під зоною для перегляду знаходиться поле, у якому користувач може залишити коментар стосовно цього шаблону і кнопка для відправки коментаря на сервер. У верхньому рядку знаходяться дві кнопки, одна для повернення назад, до списку, а інша слугує для підтвердження цього шаблону адміністратором. На вкладці Assets у верхній частині сторінки розташоване поле на всю ширину сторінки для завантаження цифрових активів, таких як зображення та відео. Під цим полем знаходиться список із вже завантажених активів. На кожному елементі цього списку знаходиться інформація, щодо того, які формати у ньому присутні, а також перелік знайдених файлів. У правій частині кожного елементу зі списку знаходиться кнопка для видалення даного активу з кампанії. На вкладці Data Sources також знаходиться поле на всю ширину сторінки для завантаження файлів з даними. Під ним розташований список з карток, що містить інформацію щодо кожного завантаженого файлу: назва, перелік знайдених колонок з даними та кнопка для видалення елементів, що були завантажені. На вкладці Audiences знаходиться сторінка для керування аудиторіями. У лівій частині сторінки під заголовком розміщено поле для пошуку аудиторій за назвами, а у правій три кнопки: створити аудиторію, видалити всі аудиторії, та імпортувати з файлу.

Якщо натиснути на кнопку імпорту аудиторій з’явиться модальне вікно для завантаження файлів. У центральній частині вікна розташована зона drag and drop до якої можна перетягнути бажаний файл, а у її центрі – кнопка, яка відкриває провідник файлової системи для вибору файлів. У правому кутку під цією зоною знаходиться кнопка завантаження. Кнопка додати аудиторію перенаправляє на сторінку створення нової аудиторії (рис. 4.4).

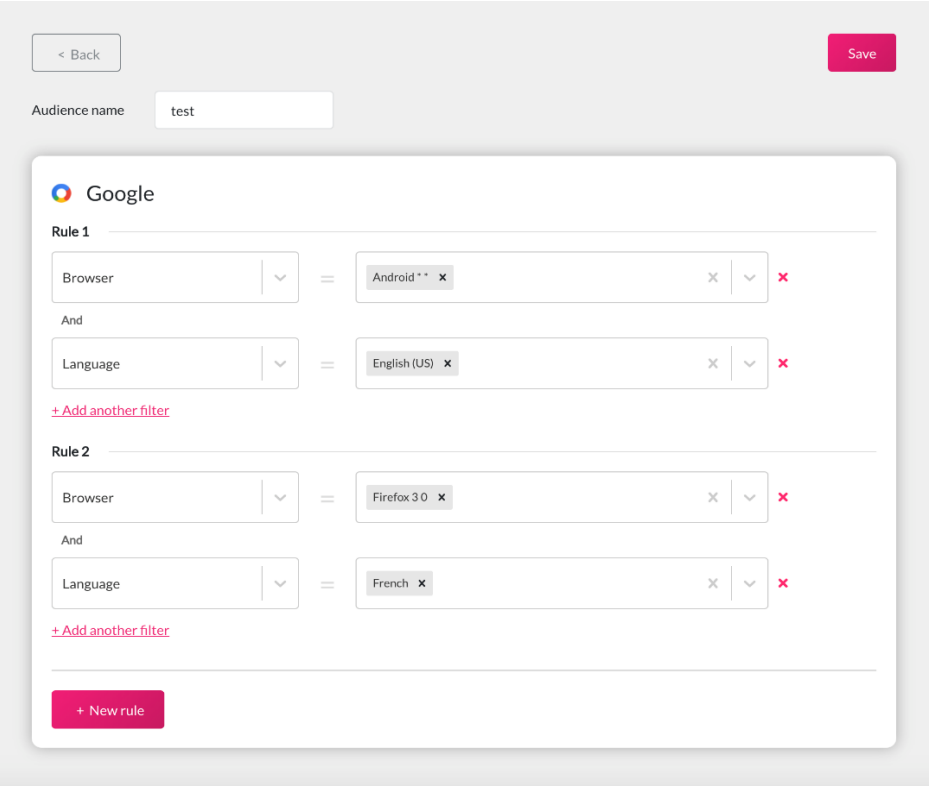


Рис. 4.4. Сторінка створення нової аудиторії

На ній міститься поле для задання назви для аудиторії, а також список полів для додавання аудиторій під кожен доступний для кампанії канал публікації. При виборі певного каналу відкривається форма з таргетингами специфічними для цієї платформи. За замовчуванням у формі показано одне правило з одним фільтром всередині. У рамках правила можна створити набір фільтрів. Фільтр представленний у вигляді пари «ключ-значення», де ключ можна вибрати із наперед заданих значень, що є специфічними для даного каналу. В залежності від обраного ключа фільтру, значення фільтру може бути або текстовим полем або випадаючим списком для вибору наперед заданих значень. Для додавання нового фільтру до правила під поточним значенням міститься кнопка додати фільтр. У правому верхньому кутку сторінки знаходиться кнопка зберегти, що збереже аудиторію зі створеними правилами та фільтрами, а також поверне користувача до сторінки керування аудиторіями. Під кнопками та полем для пошуку

розташований список аудиторій для поточної рекламної кампанії. При наведені курсором на елемент списку його фон змінюється на білий та до нього додається тінь. Кожен елемент містить назву аудиторії, та список каналів публікації для яких дана аудиторія містить правила та фільтри. Логотипи тих каналів, значення для яких містяться у аудиторію є кольоровими, а інші – сірими. Також при наведені курсору на елемент у правому куті утвореної картки з'являється кнопка видалити, для видалення поточної аудиторії з кампанії.

4.3. Тестування платформи

Щоб перевірити правильність роботи програмних засобів було створено набір тест-кейсів для проведення тестування системи.

Тест-кейс № 1

Передумови:

1. У системі створено адміністративного користувача.
2. У системі створено акаунт рекламодавця, що містить ідентифікатор рекламодавця та профіль у системі Google DCM.
3. У системі створено новий бріф, що не пов'язаний з жодною кампанією.

Сценарій тестування:

1. Увійти до системи у ролі адміністративного користувача.
2. Створити нову рекламну кампанію:
    - натиснути на кнопку «Нова кампанія» на сторінці керування кампаніями;
    - задати ім'я для нової кампанії;
    - вибрати бріф, який буде пов'язано з кампанією;
    - вибрати зі списку рекламний акаунт, що містить ідентифікатори Google DCM;

- обрати канал для публікації Google та спосіб доставки Pre-render;
- обрати початкову та кінцеву дати;
- задати конвенцію імен для місць для розміщень та реклами;
- обрати кампанію Google DCM;
- обрати сайти Google DCM;
- натиснути кнопку «Створити».

3. Завантажити шаблони: Шаблон має бути у форматі zip та містити зображення за замовчуванням з назвою backup.jpg або backup.png.
4. Створити аудиторію для каналу Google з будь-якою комбінацією таргетингів.
5. Створити набір даних хоча б для однієї варіації.
6. Схвалити створені варіації на вкладці Approval на сторінці кампанії.
7. Схвалити створені варіації на вкладці Creatives на сторінці бріфів.
8. Опублікувати варіації на сторінці перегляду кампанії, натиснувши кнопку Publish.

Очікуваний результат:

1. На сторінці перегляду кампанії має з’явитися посилання на Google Spreadsheet, а також час коли відбулась публікація.
2. У обраній DCM кампанії мають бути створені наступні сутності:
   - реклами та місця для розміщень згідно з заданими конвенціями імен;
   - реклама та рекламний креатив за замовчуванням, на основі зображення, що було завантажене у шаблоні.

Тест-кейс № 2

Передумови:

1. У системі створено адміністративного користувача.
2. У системі створено акаунт рекламодавця, що пов’язаний з акаунтом у с системі YouTube.

3. У системі створено новий бріф, що не пов’язаний з жодною кампанією.
4. Створити нову рекламну кампанію:
    - натиснути на кнопку «Нова кампанія» на сторінці керування кампаніями;
    - задати ім’я для нової кампанії;
    - вибрати бріф, який буде пов’язано з кампанією;
    - вибрати зі списку рекламний акаунт, що містить ідентифікатори YouTube;
    - обрати канал для публікації YouTube;
    - обрати початкову та кінцеву дати;
    - натиснути кнопку «Створити».
5. Завантажити шаблони: Шаблон має бути у форматі zip та містити відео у форматі mp4.
6. Створити аудиторію для каналу YouTube з будь-якою комбінацією таргетингів.
7. Створити набір даних для формату «Dynamic Video» хоча б для однієї варіації.
8. Запустити рендеринг відео-варіацій на сторінці перегляду кампанії, натиснувши кнопку «Render».
9. Схвалити створені варіації на вкладці «Approval» на сторінці кампанії.
10. Схвалити створені варіації на вкладці «Creatives» на сторінці бріфів.
11. Опублікувати варіації на сторінці перегляду кампанії, натиснувши кнопку «Publish».

Очікуваний результат:

1. У обраному YouTube акаунті, що було прив’язано до кампанії за допомогою рекламного акаунту мають з’явитися рекламні відео, що були підтверджені та опубліковані.

4.4. Рекомендації, щодо подальшого вдосконалення

Основною рекомендацією для вдосконалення платформи є інтеграція з більшою кількістю каналів публікації такими як Instagram, LinkedIn, Messenger та інших для охоплення максимальної аудиторії. Крім того варто додати інтеграцію усіх можливих типів реклами у вже існуючих каналах, наприклад: кільцева галерея та вибірка для Facebook [32] та рекламні картки для YouTube, що відображають вміст, який може бути доречним для даного відео, наприклад продукти, що розміщені на відео.

Для покращення автоматизації ведення рекламних кампаній пропонується створення алгоритму на основі моделей машинного навчання з підкріпленням. В індустрії широко використовуються такі моделі, як багаторукі бандити [33]. Інформація про попередні рекламні кампанії використовується як тренувальні дані, виходячи з яких модель обирає певні варіації креативів, вони публікуються, а модель спостерігає за успішністю рекламної кампанії та конверсією. З кожним кроком тренування модель вчиться підбирати варіації таким чином, щоб максимізувати конверсію.

Для забезпечення стабільності системи рекомендовано змінити чергу повідомлень, через нестачу її можливостей у деяких випадках. А також налагодити оркестрацію мікросервісів для стабільного функціювання та передбачуваного масштабування.

ВИСНОВКИ

Метою даного дипломного проекту було створення програмної платформи, що автоматизує процес створення та ведення рекламних кампаній, генерації та публікації рекламних креативів, а також збір статистики з різних джерел. Перед початком розроблення платформи було проведено аналіз предметної області, визначено основні проблеми, які виникають у рекламних компаній під час роботи над новими та підтримкою існуючих рекламних кампаній. Було досліджено можливі способи вирішення цих проблем та зібрано вимоги до розроблюваного програмного забезпечення. В результаті аналізу існуючих рішень було виявлено, що не існує жодної системи, яка б вирішувала ці проблеми. У пояснювальній записці наведено обґрунтування вибору архітектури системи, технологій розроблення, баз даних, а також рекламних серверів для інтеграції з ними. Значну увагу було приділено користувацькому інтерфейсу для зручного та зрозумілого користування системою.

Розроблена система дозволить забезпечити автоматизацію таких дій для рекламних компаній та агентств, як керування та створення рекламних кампаній, генерацію рекламних креативів та їх публікацію до різних медіа-майданчиків.

СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Интернет-реклама как новый инструмент для бизнеса, эффективность рекламы в интернете, размещение рекламы [Електронний ресурс]. – Режим доступу: https://www.promo-webcom.by/analytics/internet-advert/1433-internet-reklama-kak-effektivnyij-instrument-dlya/
2. Баннерная реклама | UAMASTER Digital Agency [Електронний ресурс]. – Режим доступу: https://blog.uamaster.com/media-advertising-service/
3. Контекстная реклама – Википедия [Електронний ресурс]. – Режим доступу: https://ru.wikipedia.org/wiki/Контекстная_реклама
4. Типы баннеров - Анализ баннерной рекламы как одного из основных видов интернет-маркетинга [Електронний ресурс]. – Режим доступу: https://studbooks.net/1017615/marketing/tipy_bannerov
5. Микросервисная архитектура – Википедия [Електронний ресурс]. – Режим доступу: https://ru.wikipedia.org/wiki/Микросервисная_архитектура
6. Переход от монолита к микросервисам / Хабр [Електронний ресурс]. – Режим доступу: https://habr.com/ru/post/305826/
7. Node.js – Википедия [Електронний ресурс]. – Режим доступу: https://ru.wikipedia.org/wiki/Node.js
8. NodeJS ecosystem [Електронний ресурс]. – Режим доступу: https://www.slideshare.net/Yukti_Kaura/nodejs-ecosystem-39277880
9. Overview of the JavaScript ecosystem – Kristian Poslek – Medium [Електронний ресурс]. – Режим доступу: https://medium.com/@bojzi/overview-of-the-javascript-ecosystem-8ec4a0b7a7be

10. Python – Википедия [Електронний ресурс]. – Режим доступу: https://ru.wikipedia.org/wiki/Python
11. Python vs Node.js: Which is Better for Your Project | DA-14 [Електронний ресурс]. – Режим доступу: https://da-14.com/blog/python-vs-nodejs-which-better-your-project
12. Why I’m Switching From Python To NodeJS [Електронний ресурс]. – Режим доступу: https://blog.geekforbrains.com/why-im-switching-from-python-to-nodejs-1fbc17dc797a
13. Database – Wikipedia [Електронний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/Database#Database_management_system
14. Открытая аналитическая СУБД – Arenadata documentation [Електронний ресурс]. – Режим доступу: https://docs.arenadata.io/adb/
15. Обзор - Документация ClickHouse [Електронний ресурс]. – Режим доступу: https://clickhouse.yandex/docs/ru/
16. Наш опыт внедрения ClickHouse – аналитической СУБД | DOU [Електронний ресурс]. – Режим доступу: https://dou.ua/lenta/articles/experience-with-clickhouse/
17. Vertica – Wikipedia [Електронний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/Vertica
18. HP Vertica: СУБД для анализа «больших данных» / Блог компании МУК / Хабр [Електронний ресурс]. – Режим доступу: https://habr.com/ru/company/muk/blog/249715/
19. Резидентная база данных – Википедия [Електронний ресурс]. – Режим доступу: https://ru.wikipedia.org/wiki/Резидентная_база_данных

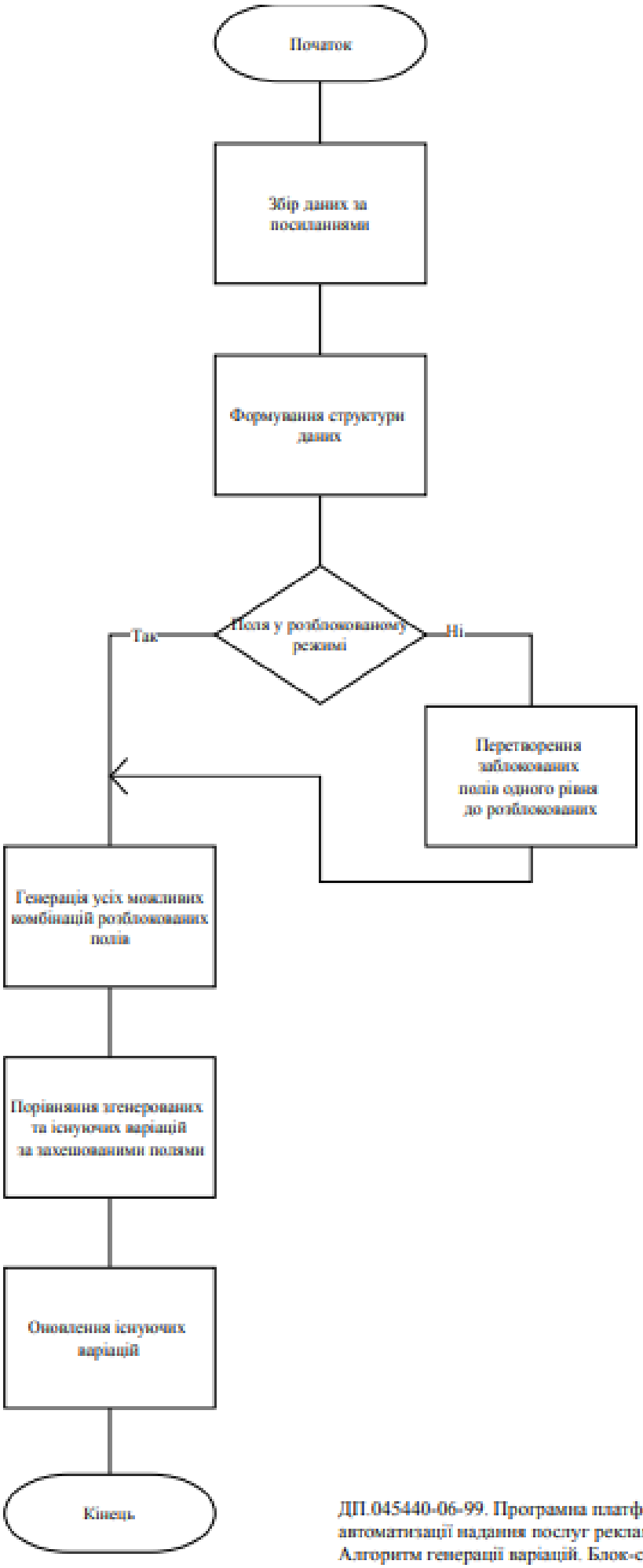
20. Redis [Електронний ресурс]. – Режим доступу: https://redis.io/documentation
21. Aerospike Documentation [Електронний ресурс]. – Режим доступу: https://www.aerospike.com/docs/
22. What Is MongoDB? | MongoDB [Електронний ресурс]. – Режим доступу: https://www.mongodb.com/what-is-mongodb
23. MongoDB vs CouchDB detailed comparison as of 2019 – Slant [Електронний ресурс]. – Режим доступу: https://www.slant.co/versus/613/14641/~mongodb_vs_couchdb
24. Apache CouchDB [Електронний ресурс]. – Режим доступу: http://couchdb.apache.org/
25. NoSQL showdown: MongoDB vs. Couchbase | JavaWorld [Електронний ресурс]. – Режим доступу: https://www.javaworld.com/article/2078750/nosql-showdown--mongodb-vs--couchbase.html
26. Overview | Campaign Manager | Google Developers [Електронний ресурс]. – Режим доступу: https://developers.google.com/doubleclick-advertisers/guides/trafficking_overview
27. Facebook Ads Manager: управление рекламой в Facebook, Instagram и т. д. | Facebook Business [Електронний ресурс]. – Режим доступу: https://ru-ru.facebook.com/business/tools/ads-manager
28. Understanding Different Types Of YouTube Video Ads | DASH TWO [Електронний ресурс]. – Режим доступу: https://dashtwo.com/blog/different-youtube-ads/
29. Пять простых шагов для понимания JSON Web Tokens (JWT) / Хабр [Електронний ресурс]. – Режим доступу: https://habr.com/ru/post/340146/

30. An introduction to Message Brokers – Xavier Geerinck – Medium [Електронний ресурс]. – Режим доступу: https://medium.com/@xaviergeerinck/an-introduction-to-message-brokers-9bd203b4ebbd
31. Выбор MQ для высоконагруженного проекта / Хабр [Електронний ресурс]. – Режим доступу: https://habr.com/ru/post/326880/
32. Характеристики рекламы с кольцевой галереей на Facebook | Руководство по рекламе на Facebook [Електронний ресурс]. – Режим доступу: https://www.facebook.com/business/ads-guide/carousel
33. Многорукие бандиты: введение и алгоритм UCB1 / Блог компании Surfingbird / Хабр Facebook [Електронний ресурс]. – Режим доступу: https://habr.com/ru/company/surfingbird/blog/168611/

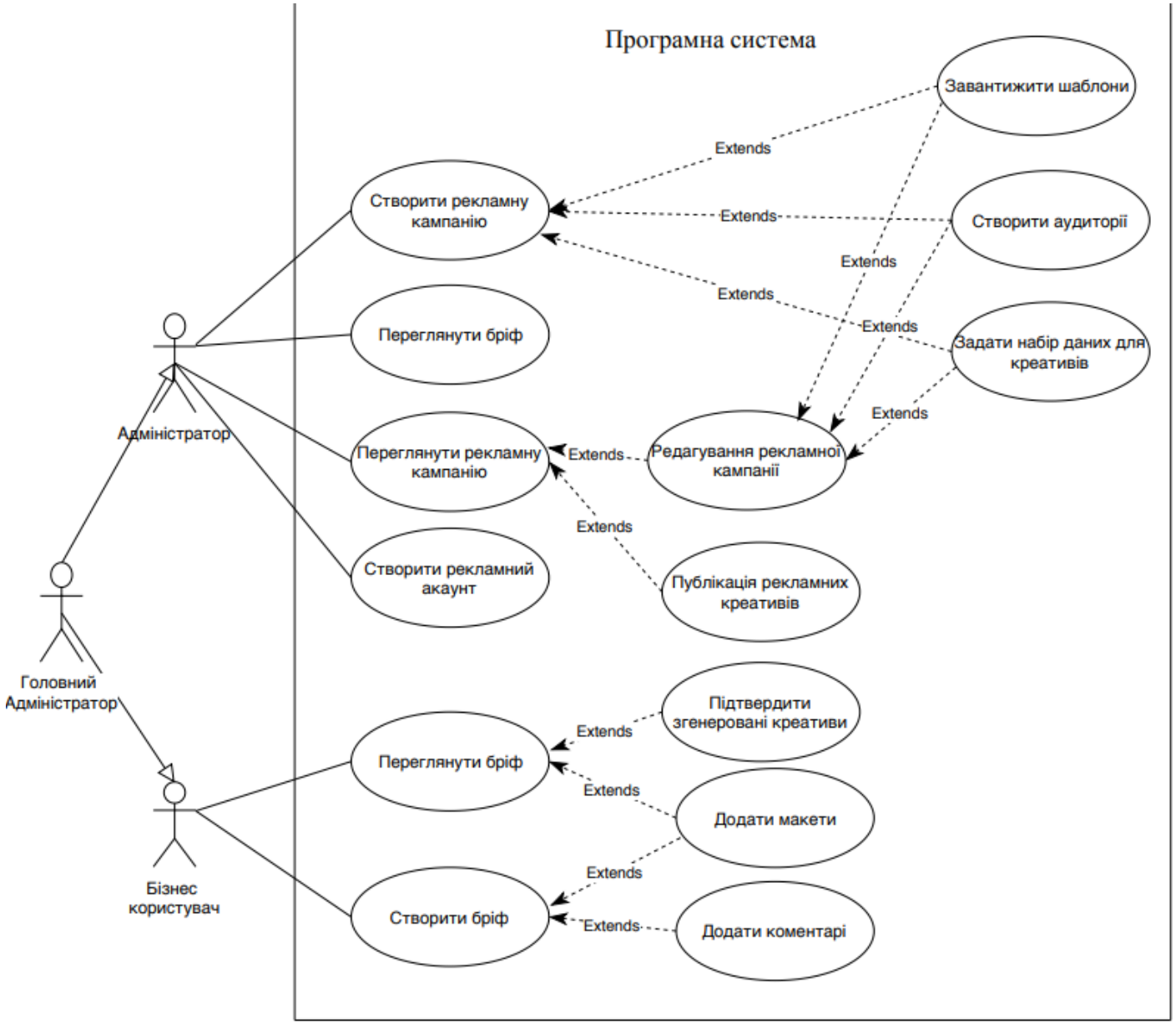
ДОДАТКИ

Додаток 1

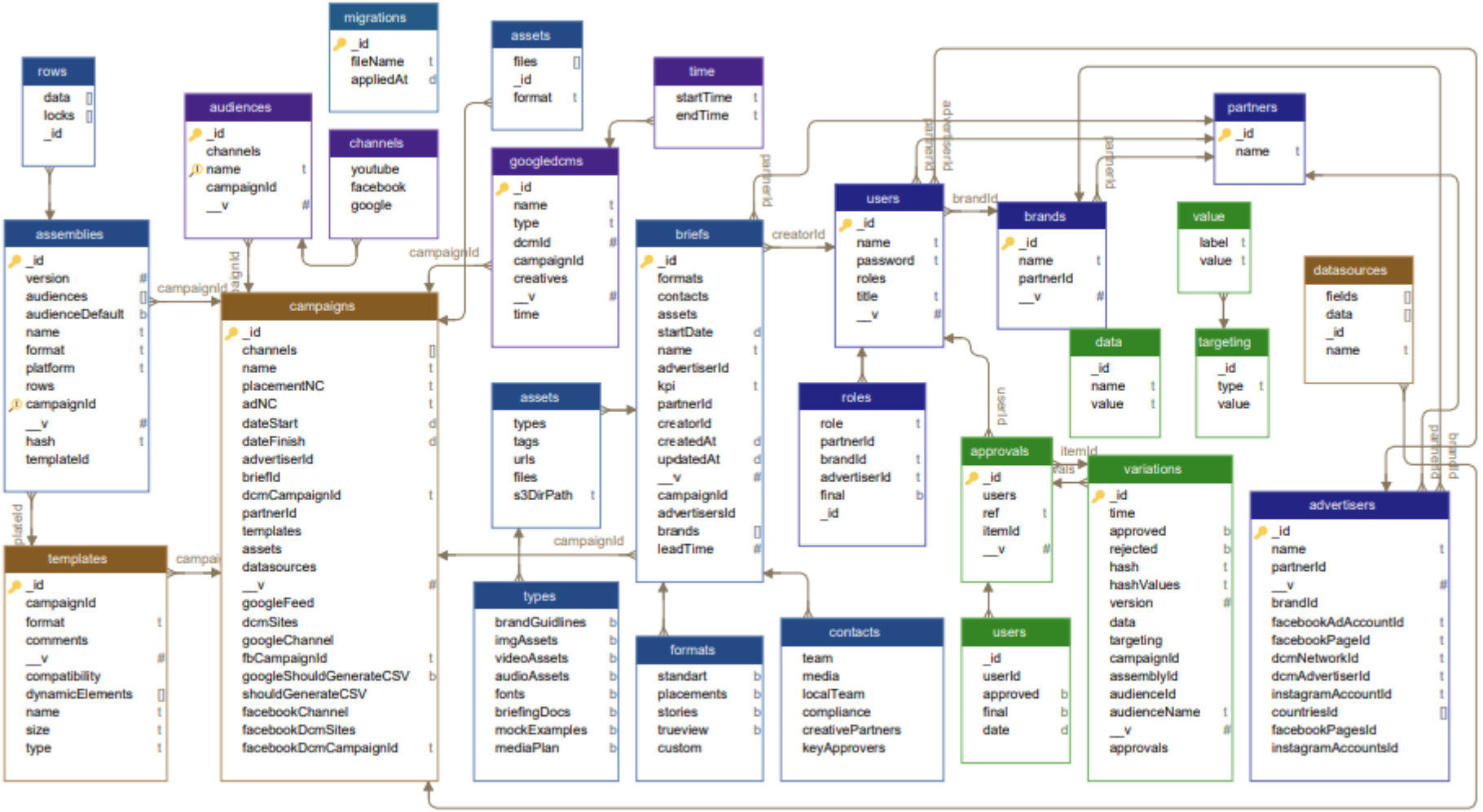
Копії графічних матеріалів



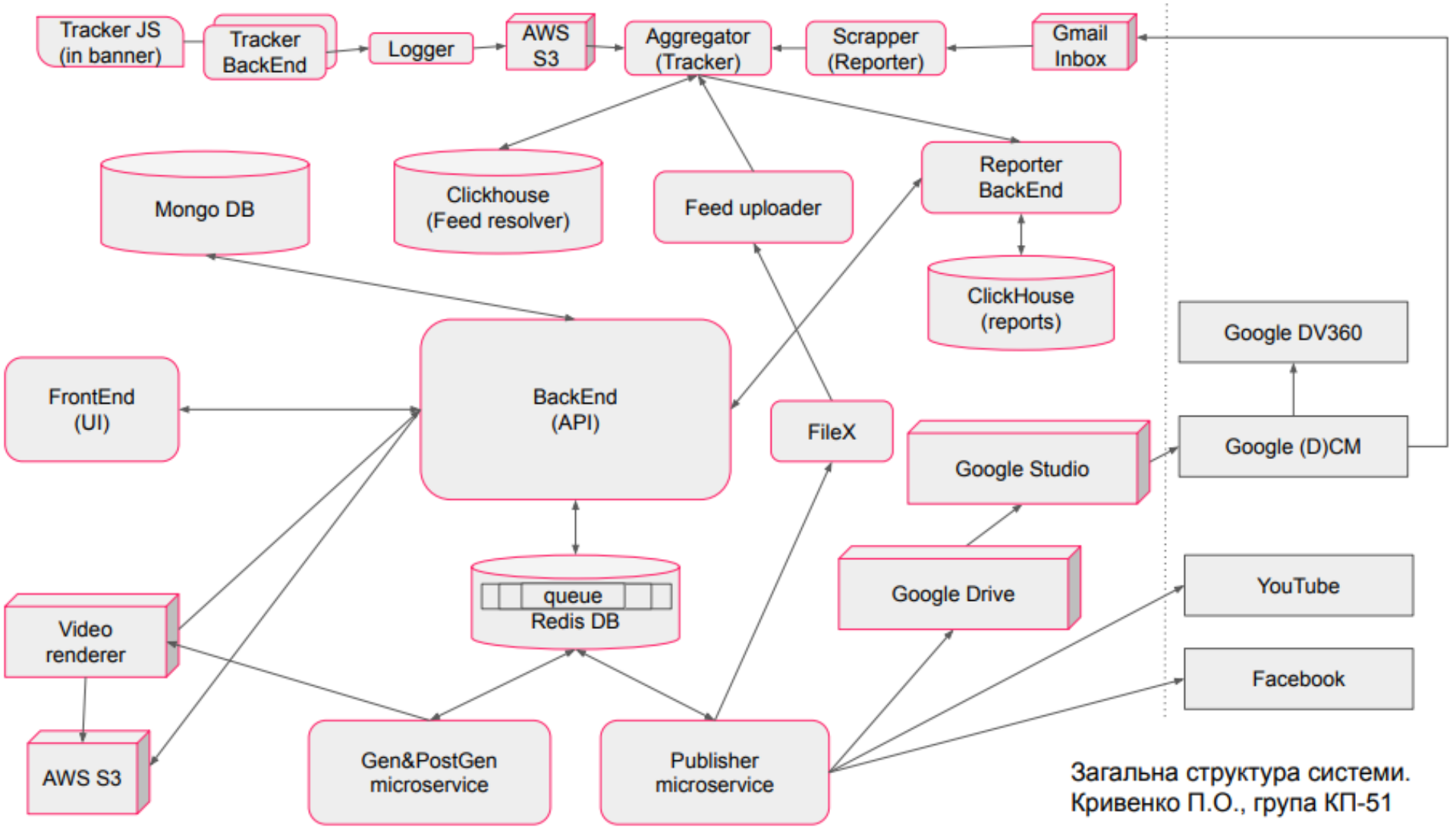
ДП.045440-06-99. Програмна платформа для автоматизації надання послуг рекламним компаніям. Алгоритм генерації варіацій. Блок-схема



ДП.045440-07-99. Програмна платформа для автоматизації надання послуг рекламним компаніям. Варіанти використання. Діаграма прецедентів



Структура бази даних. Кривенко П.О., група КП-51



Загальна структура системи.
Кривенко П.О., група КП-51

Додаток 2

Лістинг мікросервісу генерації варіацій

index.js

```
require('dotenv').config();
require('./log');
const db = require('db');

const { variationGenerator, variationPublisher } = require('core').queue;
const { generateByAssembly } = require('./dal');
const videoQueueProcessor = require('./queue/videoQueueProcessor');

const { MONGO_CONNECTION, REDIS_CONNECTION } = require('./config');

db.connect(`mongodb://${MONGO_CONNECTION.host}:${MONGO_CONNECTION.port}/${M
ONGO_CONNECTION.db}`, {
  useCreateIndex: true,
  useNewUrlParser: true,
  useFindAndModify: false,
  reconnectInterval: 10000, // 10 seconds
  reconnectTries: Number.MAX_VALUE, // reconnect retry forever
});
db.connection.on('error', console.error.bind(console, 'connection
error:'));

const queue = new variationGenerator.Queue(REDIS_CONNECTION);
const variationPublisherQueue = new
variationPublisher.Queue(REDIS_CONNECTION);

queue.process({
  onMessageGenerate: async (msg, done) => {
    try {
      const forceRegenerate = !!msg.force;
      const updatedAssemblies = await Promise.all(
        msg.assemblies.map(assembly => generateByAssembly(msg.campaignId,
assembly, forceRegenerate)),
      );

      const updated = updatedAssemblies.reduce((acc, res) => acc || res,
false);

      if (updated) {
        // if at least one assembly was updated - publish campaign
        console.log(`Campaign was updated. Sent to Publisher. campaignId
${msg.campaignId}`);
        variationPublisherQueue.publish(
          new variationPublisher.MessagePublish({ campaignId:
msg.campaignId }),
        );
      }
      done(null, true);
    } catch (e) {
      console.log('onMessageGenerate exception caught');
      console.log(e);
      done(e);
    }
  },
});

videoQueueProcessor.initProcessing();

console.log('Generator service started');
```

dal.js

```
const { isTimeOfTheDay, variationsPermutation, formats } =
require('common');

const {
  Audiences, Campaigns, Assemblies, Variations, Approvals, Templates,
} = require('db').models;

const { getHash } = require('./utils/utils');

/**
 * prepareVariations
 * prepare variation for storing
 * variation: {items, audience, audienceFilters}
 * @param {*} generatedVariations list of variations received from
permutation function
 * @param {*} campaignId campaignId
 * @param {*} assembly assembly
 */
function prepareVariations(generatedVariations, campaignId, assembly) {
  return generatedVariations.map(variation => {
    const varItems = Object.keys(variation.items).map(key => ({
      name: key,
      value: variation.items[key],
    }));

    // calculate hash of the content
    const mergedVarItems = varItems.reduce((acc, item) =>
`${acc}${item.name} - ${item.value}`, '');
    const hashValues = getHash(mergedVarItems);

    // calculate complete variation hash
    const audienceId = (variation.audience &&
variation.audience._id.toString()) || '';
    const platformAndFormat = `${assembly.platform}-${assembly.format}`;

    const hash = getHash(`${audienceId}-${hashValues}-
${platformAndFormat}`);

    // find and extract DOW/TOD
    const varTime = variation.audienceFilters.find(filter =>
isTimeOfTheDay(filter.filterType));

    // convert audiences to targeting, just avoid filter naming
    const targeting = variation.audienceFilters.map(filter => ({
      type: filter.filterType,
      value: filter.filterValue,
    }));

    return {
      hash,
      hashValues,
      version: assembly.version,
      data: varItems,
      targeting,
      campaignId,
      assemblyId: assembly._id,
      ...(varTime && { time: varTime.filterValue }),
      ...(variation.audience && { audienceId: variation.audience._id }),
      ...(variation.audience && { audienceName: variation.audience.name }),
    };
  });
}

/**
```

```
 * getVariationsWithSameHashes
 */
async function getCurrentVariationsWithSameHashes(variations) {
  const hashes = variations.map(v => v.hash);

  // find variations with the hashes in DB
  const allVariations = await Variations.find({
    hash: { $in: hashes },
  });

  // sort and filter for the latest version
  const versionHashes = allVariations.reduce((acc, val) => {
    if (acc[val.hash]) {
      if (acc[val.hash].version < val.version) {
        acc[val.hash] = val;
      }
    } else {
      acc[val.hash] = val;
    }
    return acc;
  }, {});

  return Object.values(versionHashes);
}

/**
 * insertIncomingVariations
 * copy approvals if there is variation with same hash
 */
async function insertIncomingVariations(incomingVariations,
currentVariationsWithSameHashes) {
  return Promise.all(
    incomingVariations.map(async incomingVariation => {
      const currentVariation = currentVariationsWithSameHashes.find(
        current => current.hash === incomingVariation.hash,
      );

      if (currentVariation) {
        // update approved/rejected
        incomingVariation.approved = currentVariation.approved;
        incomingVariation.rejected = currentVariation.rejected;

        // updated approvals
        incomingVariation.approvals = currentVariation.approvals;
        incomingVariation.job = currentVariation.job;
      }

      const variation = new Variations(incomingVariation);
      const savedVariation = await variation.save();

      if (currentVariation) {
        // update Approvals db with updated id
        await Approvals.updateOne(
          { _id: currentVariation.approvals },
          { itemId: savedVariation._id },
        );
      }
    }),
  );
}

/**
 * Generate variations for a single assembly
 * @param {*} campaignId campaign id
```

```
 * @param {*} assemblyId assembly id
 * @returns true if generation is successful and false if generation was
skipped
 */
async function generateByAssembly(campaignId, assemblyId) {
  // calculate performance
  const timeStart = new Date();

  // increase version
  const assembly = await Assemblies.findOneAndUpdate(
    { _id: assemblyId },
    { $inc: { version: 1 } },
    {
      new: true,
      lean: true,
    },
  );

  const template = assembly.templateId
    ? await Templates.findOne({ _id: assembly.templateId }, null, { lean:
true })
    : null;

  const audiences = await Audiences.find({ _id: { $in: assembly.audiences }
}, null, {
    lean: true,
  });

  const campaign = await Campaigns.findOne(
    { _id: campaignId },
    { datasources: true },
    { lean: true },
  );

  const dynamicElements = formats.getDynamicElements(assembly.format,
template);

  // STEP 1. Execute permutations
  const generatedVariations = variationsPermutation(
    assembly,
    audiences,
    dynamicElements,
    campaign.datasources,
  );

  // STEP 2. Format variations and prepare for DB
  const incomingVariations = await prepareVariations(generatedVariations,
campaignId, assembly);

  // STEP 3. Find variations with the same hashes in DB
  const currentVariationsWithSameHashes = await
getCurrentVariationsWithSameHashes(
    incomingVariations,
  );

  // STEP 4. Insert into DB incoming variations
  await insertIncomingVariations(incomingVariations,
currentVariationsWithSameHashes);

  const timeFinish = new Date();
  console.log(
    `Inserted: ${incomingVariations.length} vars; ${timeFinish
      - timeStart}ms; assemblyId: ${assemblyId}`,
  );
```

```
}

async function generateAll() {
  // calculate performance
  const timeStart = new Date();
  const assemblies = await Assemblies.find({}, { campaignId: true });
  /* eslint-disable no-restricted-syntax */
  if (assemblies.length > 0) {
    try {
      const BULK_AMOUNT = 50;
      const bulks = [];
      for (let i = 0; i < assemblies.length / BULK_AMOUNT; i += 1) {
        bulks.push(assemblies.slice(i * BULK_AMOUNT, (i + 1) *
BULK_AMOUNT));
      }
      for (let i = 0; i < bulks.length; i += 1) {
        const bulk = bulks[i];
        console.log('-------------------------------------');
        console.log(`Proicessing bulk #${i + 1}/${bulks.length}; bulk =
${bulk.length}`);
        const memBefore = Math.round(process.memoryUsage().heapUsed / 1024
/ 1024);

        // eslint-disable-next-line no-await-in-loop
        await Promise.all(
          bulk.map(async assembly => {
            try {
              await generateByAssembly(assembly.campaignId, assembly._id,
false);
            } catch (e) {
              console.log('Exception caught', e.toString(), `assemblyId:
${assembly._id}`);
            }
          }),
        );
        const memAfter = Math.round(process.memoryUsage().heapUsed / 1024 /
1024);
        const total = Math.round(process.memoryUsage().heapTotal / 1024 /
1024);
        console.log(
          `Memory stats. Bulk #${i
            + 1}. Before ${memBefore} Mb; After: ${memAfter} Mb; Total
Heap: ${total} Mb`,
        );
      }
    } catch (e) {
      console.log('Exception caught in generateByAssembly', e.toString());
    }
  }
  const timeFinish = new Date();
  console.log(`Generation Cron. ${assemblies.length} assemblies in
${timeFinish - timeStart}ms`);
}

module.exports = { generateByAssembly, generateAll, prepareVariations };
```

videoQueueProcessor.js

```
const { videoGenerator } = require('core').queue;

const { setVariationUpdate } = require('core').dals.variation;
```

```
const { REDIS_CONNECTION } = require('../config');

const videoQueue = new videoGenerator.Queue(REDIS_CONNECTION);

function initProcessing() {
  videoQueue.process({
    onMessageGetVideoToRender: (jobId, variationId, done) => {
      try {
        setVariationUpdate(variationId, { id: jobId, status: 'new' });
        console.log(`Variation ${variationId} added to rendering queue`);
      } catch (e) {
        console.log('onMessageGetVideoToRender exception caught');
        console.log(e);
        done(e);
      }
    },
  });
}

module.exports = { initProcessing };
```

Додаток 3

Лістинг мікросервісу публікації креативів

google/BaseGooglePublisher.js

```
/* eslint-disable */
const axios = require('axios');
const moment = require('moment');
const config = require('./config');
const zlib = require('zlib');
const fs = require('fs');
const util = require('util');
const gzip = util.promisify(zlib.gzip);

const { writeFile } = require('core').fs;

const log = require('../../utils/logger');

const { REPORTER_STORAGE_PATH, FILEX_ENDPOINT } = config;

const GoogleDriveService = require('../../services/GoogleDriveService');
const generateCSV = require('./utils/generateCsv');
const { googleFieldTypes, googleFields, channels, isLocation,
isTimeOfTheDay } = require('common');
const { getAllLatestVersionVariations, isActiveTime } =
require('core').dals.variation;
const { Campaigns, Partners, Advertisers } = require('db').models;

const AssetUrlConfig = {
  S3_REGION: process.env.S3_REGION,
  S3_UPLOAD_BUCKET: process.env.S3_UPLOAD_BUCKET,
};

class BaseGooglePublisher {
  constructor(campaign) {
    this.campaign = campaign;
    this.preparedVariations = {};
  }

  getAllColumns(defaultColumns) {
    throw new Error('Must be implemented by child');
  }

  prepareData(){
    // Should be implemented
  }

  async publish(variations) {
    log.info(`Google publish`);
    const preparedVariations = await this.prepareData(variations);
    this.preparedVariations = preparedVariations.variations;
    const data = this.buildData(preparedVariations.CSV);

    const columns = this.getAllColumns(preparedVariations.CSV);
    const csv = this.generateCSV(columns, data);
    return Promise.all([this.publishCSV(csv),
this.generateCSVForReporter(this.campaign._id)]);
  }

  generateCSV(columns, data) {
    log.info(`CSV generation`);
    return generateCSV(columns, data);
  }

  buildData(variations) {
```

```
    log.info(`Building data`);
    const campaignId = this.campaign._id;
    const data = variations.map((variation, index) => {
      const items = variation.data.reduce((acc, item) => {
        let assetsUrl = null;
        if (['image', 'video'].includes(googleFieldTypes[item.name])) {
          assetsUrl =
`https://s3.${AssetUrlConfig.S3_REGION}.amazonaws.com/${
            AssetUrlConfig.S3_UPLOAD_BUCKET

}/${campaignId}/assets/${variation.assembly.format}/${item.value}`;
        }
        acc[item.name] = assetsUrl || item.value;
        return acc;
      }, {});

      const isDefault = variation.assembly.audienceDefault ? 'TRUE' :
'FALSE';

      const baseItems = {
        id: index + 1,
        Reporting_Label: `${campaignId}-${variation.hash}`,
        Active: isActiveTime(variation) ? 'TRUE' : 'FALSE',
        Weights: 1,
        isDefault,
        FORMAT: variation.assembly.format,
      };

      return { ...baseItems, ...items };
    });
    return data;
  }

  async publishCSV(csv) {
    log.info(`CSV publishing`);
    const { name, googleFeed } = this.campaign;
    const fileId = googleFeed && googleFeed.fileId;
    const gs = new GoogleDriveService();
    const fileName = name;

    try {
      const resultFileId = await gs.syncData({
        csv,
        fileName,
        folderId: process.env.GOOGLE_FEED_FOLDER_ID,
        ...(fileId && { fileId }),
      });
      return resultFileId;
    } catch (e) {
      log.info(`CSV publishing error: ${e}`);
      return;
    }
  }

  getVariationTargeting(variation) {
    const targeting = variation.targeting
      //      .filter(target => !isTimeOfTheDay(target.type))
      .map(target => {
        let value = target.value;
        if (isLocation(target.type)) {
          // if location than value is an array of locations
          value = target.value.reduce(
            (res, location, index) => res + (index > 0 ? '|' : '') +
JSON.parse(location.value)['Canonical Name'],
```

```
            '',
          );
        }

        return {
          name: target.type,
          value,
        };
      });

    return targeting;
  }

  getAllColumnsForReporter(variations, defaultColumns) {
    const allColumns = variations.reduce((acc, variation) => {
      const targeting = this.getVariationTargeting(variation).filter(t =>
!isTimeOfTheDay(t.name));
      const dataWithTargeting = [...variation.data, ...targeting];

      const columnNames = dataWithTargeting.map(item => item.name);
      return [...acc, ...columnNames];
    }, defaultColumns);

    return [...new Set(allColumns)];
  }

  get getFieldsForReporter() {
    return [];
  }

  async generateCSVForReporter(campaignId, additionalData) {
    log.info(`Publish CSV Reporter: campaignId ${campaignId}`);
    const campaign = await Campaigns.findById(campaignId);

    const variations = await getAllLatestVersionVariations(campaignId,
channels.google);

    const mainFields = this.getFieldsForReporter;

    const columns = this.getAllColumnsForReporter(variations,
[...mainFields, ...googleFields]);
    const partner = await Partners.findOne({ _id: campaign.partnerId });
    const advertiser = await Advertisers.findOne({ _id:
campaign.advertiserId });

    const data = variations.map(variation => {
      const mainData = {
        campaignId: campaignId.toString(),
        campaignName: campaign.name,
        partnerId: partner._id.toString(),
        partnerName: partner.name,
        advertiserId: advertiser._id.toString(),
        advertiserName: advertiser.name,
        // NOTE [SZ]. variationId set to be variation hash as a temporary
migration solution.
        // When all received reports are new - variationId field can be
restored/removed
        variationId: variation.hash,
        variationHash: variation.hash,
        audienceId: variation.audienceId ? variation.audienceId.toString()
: '',
        audienceName: variation.audienceName,
        isDefault: variation.assembly.audienceDefault ? 'TRUE' : 'FALSE',
        format: variation.assembly.format,
```

```
      };
      if (additionalData) {
        const keys = Object.keys(additionalData[variation.hash]);
        keys.forEach(key => {
          mainData[key] = additionalData[variation.hash][key];
        });
      }

      //  allValues array of [{name: "", value: ""}]
      //
      const allValues = [...variation.data,
...this.getVariationTargeting(variation)];

      const values = allValues.reduce((acc, item) => {
        acc[item.name] = item.value;
        return acc;
      }, {});

      return { ...mainData, ...values };
    });

    // generate
    const generatedCSV = this.generateCSV(columns, data);

    if (!variations || variations.length === 0) {
      log.info(`Publish CSV Reporter finished. No variations found.
campaignId ${campaignId}`);
      return false;
    }

    //    await writeFile(`google_feeds_v1_${campaignId}-
${moment().format('YYYYMMDD-hhmmss')}.csv`, generatedCSV);

    const reporterFileName = `google_feeds_v1_${campaignId}-
${moment().format('YYYYMMDD-hhmmss')}.csv.gz`;
    await
writeFile(`${process.env.REPORTER_STORAGE_PATH}/${reporterFileName}`, await
gzip(generatedCSV));

    // TODO: handle results of this
    if (FILEX_ENDPOINT) {
      await axios.post(`${FILEX_ENDPOINT}`, {
        file: {
          path: `file://${reporterFileName}`,
          group: 'GOOGLE_FEEDS',
          host: 'myhostname',
        },
      });
    }

    campaign.shouldGenerateCSV[channels.google] = false;
    await campaign.save();

    log.info(`Publish CSV Reporter finished. CSV file saved successfully.
campaignId ${campaignId}`);
    return true;
  }
}

module.exports = BaseGooglePublisher;
```

google/DCMTraffickedGooglePublisher.js

```
/* eslint-disable */
const fs = require('fs');
const fse = require('fs-extra');
const BiMap = require('bidirectional-map');
const moment = require('moment');
const _ = require('lodash');
const GoogleDCMService = require('core/src/services/GoogleDCMService');
const BaseGooglePublisher = require('./BaseGooglePublisher');
const log = require('../../utils/logger');
const { models } = require('db');
const { isTimeOfTheDay } = require('common');
const { downloadAndUnpackStuff } =
require('../../services/DCMCreative/publisher');

const { Advertisers, GoogleDCM, Templates } = models;

const defaultColumns = ['id', 'Reporting_Label', 'Active', 'Weights',
'isDefault', 'FORMAT'];
const adNCBiMap = new BiMap({
  format: '@{format}',
  'Connection Type': '@{connection_type}',
  'Operating Systems': '@{operating_systems}',
  Browser: '@{browser}',
  'Platform Type': '@{platform_type}',
  'Mobile carrier': '@{mobile_carrier}',
  Language: '@{language}',
  Group: '@{group}',
});

const placementNCBiMap = new BiMap({
  format: '@{format}',
  'Connection Type': '@{connection_type}',
  'Operating Systems': '@{operating_systems}',
  Browser: '@{browser}',
  'Platform Type': '@{platform_type}',
  'Mobile carrier': '@{mobile_carrier}',
  Language: '@{language}',
  Group: '@{group}',
});

const adTargetingsMap = new BiMap({
  'Connection Type': 'connectionTypes',
  Browser: 'browsers',
  'Platform Type': 'platformTypes',
  'Mobile carrier': 'mobileCarriers',
  'Operating Systems': 'operatingSystems',
  Language: 'languages',
});

const placementDefaultMacroses = ['@{format}', '@{site}'];
const adDefaultMacroses = ['@{format}'];
const linkingMacros = [
  '@{format}',
  '@{connection_type}',
  '@{operating_systems}',
  '@{browser}',
  '@{platform_type}',
  '@{mobile_carrier}',
  '@{language}',
  '@{group}',
];

class DCMTraffickedGooglePublisher extends BaseGooglePublisher {
  constructor(campaign) {
```

```
    super(campaign);
    this.siteIds = this.campaign.dcmSites || [];
  }

  hashCode(s) {
    return s.split('').reduce(function(a, b) {
      a = (a << 5) - a + b.charCodeAt(0);
      return a & a;
    }, 0);
  }

  groupIntoAds(variations, namingConvention, fieldsToLink) {
    const fields = namingConvention.match(/\@\{[a-zA-Z]+_?[a-zA-
Z]+\}/g).reduce((acc, macro) => {
      const field = adNCBiMap.getKey(macro);
      if (field) acc.push(field);
      return acc;
    }, []);

    const groups = {};
    variations.forEach(variation => {
      const variationTargetingMap = new Map(variation.targeting.map(obj =>
[obj.type, obj.value]));

      const key = fields.reduce(
        (acc, field) =>
          field === 'format'
            ? acc.replace(adNCBiMap.get(field), variation.assembly.format)
            : acc.replace(adNCBiMap.get(field),
this.getValueForMacrosReplacement(field, variationTargetingMap)),
        namingConvention,
      );

      if (!groups[key]) {
        const valuesToLink = fieldsToLink.reduce((acc, field) => {
          if (field === 'format') acc.push(variation.assembly.format);
          else if (!variation.assembly.audienceDefault)
acc.push(JSON.stringify(variationTargetingMap.get(field)));
          return acc;
        }, []);
        const hash = valuesToLink.reduce((acc, value) => value ? acc +
this.hashCode(value) : acc, '');
        groups[key] = {
          params: {
            hash,
            targeting: variation.targeting,
            format: variation.assembly.format,
          },
          variations: [],
        };
      }
      groups[key].variations.push(variation);
    });

    return groups;
  }

  getDataForDs(dsObject) {
    const fileName = dsObject.name;
    const ds = this.campaign.datasources.find(d => d.name === fileName);
    const usedFields = dsObject.columns;

    const res = usedFields.reduce((acc, field) => {
      const colIndex = ds.fields.indexOf(field);
```

```
      acc[field] = ds.data.map(row => row[colIndex]);
      return acc;
    }, {});

    return {
      [fileName]: res,
    };
  }

  handleOneToOneConn(fileData) {
    const fileName = Object.keys(fileData)[0];

    const dataObj = fileData[fileName];
    const keys = Object.keys(dataObj);
    const combinations = fileData[fileName][keys[0]].map((val, index) => {
      return keys.reduce((acc, key) => {
        acc[`${fileName}.${key}`] = fileData[fileName][key][index];
        return acc;
      }, {});
    });
    return combinations;
  }

  shuffle(files) {
    if (files.length < 2) return files;
    const [comb1, comb2, ...rest] = files;
    const combinations = [];

    for (let i = 0; i < comb1.length; i++) {
      for (let j = 0; j < comb2.length; j++) {
        combinations.push({ ...comb1[i], ...comb2[j] });
      }
    }
    return this.shuffle([combinations, ...(rest || [])]);
  }

  generatePlacements(variations, namingConvention, fieldsToLink) {
    const pairs = placementNCBiMap;
    const fields = namingConvention.match(/\@\{[a-zA-Z]+_?[a-zA-
Z]+\}/g).reduce((acc, macro) => {
      const field = pairs.getKey(macro);
      if (field) {
        acc.push(field);
      }
      return acc;
    }, []);
    const usedDatasources = (namingConvention.match(/\$\{.+?\}/g) || [])
      .map(macro => macro.substring(2, macro.length - 1))
      .reduce((acc, el) => {
        const fileName = el.substring(0, el.lastIndexOf('.'));
        const columnName = el.substring(el.lastIndexOf('.') + 1,
el.length);
        const item = acc.find(e => e.name === fileName);
        if (!item) {
          acc.push({
            name: fileName,
            columns: [columnName],
          });
        } else {
          item.columns.push(columnName);
        }
        return acc;
      }, []);
```

```
    const dsKeyVal = usedDatasources.map(ds => this.getDataForDs(ds));
    const normilizedFiles = dsKeyVal.map(el =>
this.handleOneToOneConn(el));
    const combinations = this.shuffle(normilizedFiles).length ?
this.shuffle(normilizedFiles)[0] : [];
    const groups = {};
    variations.forEach(variation => {
      const variationTargetingMap = new Map(variation.targeting.map(obj =>
[obj.type, obj.value]));
      const keySitesTemplates = Array.from({ length: this.siteIds.length },
(el, index) => ({
        key: namingConvention.replace('@{site}', this.siteIds[index]),
        params: { siteId: this.siteIds[index] },
      }));

      const keysTemplates = combinations.length
        ? combinations.reduce((acc, combination) => {
            keySitesTemplates.forEach(({ key, params }) => {
              const valuesToRaplace = Object.keys(combination);
              valuesToRaplace.forEach(value => (key = key.replace('${' +
value + '}', combination[value])));
              acc.push({ key, params });
            });
            return acc;
          }, [])
        : [...keySitesTemplates];

      const keys = fields.reduce(
        (acc, field) => {
          return (acc = acc.map(({ key, params }) => ({
            key: key.replace(
              pairs.get(field),
              field === 'format'
                ? variation.assembly.format
                : this.getValueForMacrosReplacement(field,
variationTargetingMap),
            ),
            params,
          })));
        },
        [...keysTemplates],
      );

      keys.forEach(({ key, params }) => {
        if (!groups[key]) {
          const { format } = variation.assembly,
            [width, height] = format.split('x');

          const valuesToLink = fieldsToLink.reduce((acc, field) => {
            if (field === 'format') acc.push(variation.assembly.format);
            else if (!variation.assembly.audienceDefault)
acc.push(JSON.stringify(variationTargetingMap.get(field)));
            return acc;
          }, []);
          const hash = valuesToLink.reduce((acc, value) => value ? acc +
this.hashCode(value) : acc, '');

          groups[key] = {
            params: {
              hash,
              ...params,
              format,
              size: { width, height },
            },
```

```
          };
        }
      });
    });

    return groups;
  }

  getMacrosesToAdd(defaultMacroses, namingConvention) {
    return defaultMacroses.reduce((acc, macros) => {
      if (!namingConvention.includes(macros)) acc.push(macros);
      return acc;
    }, []);
  }

  getValueForMacrosReplacement(field, targetingMap) {
    const targetingArray = targetingMap.get(field);
    if (field === 'Group') return targetingArray || 'ANY';

    const value = targetingArray ? targetingArray.map(v =>
v.label).join('&') : 'ANY';
    return value;
  }

  getFinalNamingConvencion(namingConvention, macrosesToAdd) {
    return macrosesToAdd.length ? namingConvention.concat('_',
macrosesToAdd.join('_')) : namingConvention;
  }

  getLinkingFieldsIntersection(placementNC, adNC) {
    const placementLinkingFields = linkingMacros.reduce((acc, macros) => {
      const placementFieldFromMacros = placementNCBiMap.getKey(macros),
        adFieldFromMacros = adNCBiMap.getKey(macros);
      if (
        placementNC.includes(macros) &&
        adNC.includes(macros) &&
        placementFieldFromMacros &&
        adFieldFromMacros &&
        placementFieldFromMacros === adFieldFromMacros
      )
        acc.push(placementNCBiMap.getKey(macros));
      return acc;
    }, []);
    return placementLinkingFields;
  }

  getAllColumns(variations) {
    const allColumns = variations.reduce((acc, variation) => {
      const columnNames = variation.data.map(item => item.name);
      return [...acc, ...columnNames];
    }, defaultColumns);
    return [...new Set(allColumns)];
  }

  async getCreativesForPairing(dcm) {
    const loadedCreatives = await
dcm.getCampaignCreatives(this.campaign._id);
    const linkedCreatives = await
dcm.getCampaignLinkedCreatives(this.campaign.dcmCampaignId);
    const creatives = loadedCreatives.data.creatives;
    const excludeCreatives =
linkedCreatives.data.campaignCreativeAssociations.map(c => c.creativeId);

    return { creatives, excludeCreatives };
```

```
  }

  async applyCreatives(dcm, creatives, excludeCreatives, advert, formats) {
    for (let creative of creatives) {
      if (!excludeCreatives.includes(creative.id)) {
        await dcm.linkCreativeToCampaign(creative.id,
this.campaign.dcmCampaignId);
      }
    }
    const appliableCreatives = creatives.filter(c =>
formats.includes(c.size.width + 'x' + c.size.height));

    if (appliableCreatives.length) {
      advert.creativeRotation = {};
      advert.creativeRotation.creativeAssignments =
appliableCreatives.reduce((acc, creative) => {
        acc.push({
          ...creative,
          creativeId: creative.id,
          creativeIdDimensionValue: creative.idDimensionValue,
          active: true,
          startTime: advert.startTime,
          clickThroughUrl:
            (creative.clickTags && creative.clickTags[0].clickThroughUrl)
||
            (creative.counterCustomEvents &&
creative.counterCustomEvents[0].exitClickThroughUrl) ||
            (creative.exitCustomEvents &&
creative.exitCustomEvents[0].exitClickThroughUrl) ||
            (creative.timerCustomEvents &&
creative.timerCustomEvents[0].exitClickThroughUrl) ||
            creative.backupImageClickThroughUrl ||
            (creative.backupImageExit &&
creative.backupImageExit.exitClickThroughUrl),
        });

        return acc;
      }, []);
      advert.active = true;
    }
  }

  async prepareData(variations) {
    const placementMacrosesToAdd =
this.getMacrosesToAdd(placementDefaultMacroses, this.campaign.placementNC);
    const dynamicDefaultMacroses = variations.reduce((acc, variation) => {
      variation.targeting.forEach(({ type }) => {
        const macro = adNCBiMap.get(type);
        if (!acc.includes(macro)) acc.push(macro);
      });
      return acc;
    }, []);

    const adMacrosesToAdd =
this.getMacrosesToAdd(adDefaultMacroses.concat(dynamicDefaultMacroses),
this.campaign.adNC);

    const placementNamingConvention =
this.getFinalNamingConvencion(this.campaign.placementNC,
placementMacrosesToAdd);
    const adNamingConvention =
this.getFinalNamingConvencion(this.campaign.adNC, adMacrosesToAdd);
```

```
    const linkingFieldsIntersection =
this.getLinkingFieldsIntersection(placementNamingConvention,
adNamingConvention);

    const adGroups = this.groupIntoAds(variations, adNamingConvention,
linkingFieldsIntersection);

    const placements = this.generatePlacements(variations,
placementNamingConvention, linkingFieldsIntersection);
    const { dcmAdvertiserId, dcmNetworkId } = await
Advertisers.findById(this.campaign.advertiserId);
    const dcm = new GoogleDCMService({ profileId: dcmNetworkId,
dcmAdvertiserId });
    const dcmCampaign = (await
dcm.getCampaign(this.campaign.dcmCampaignId)).data;

    const createdPlacements = await GoogleDCM.find({ type: 'PLACEMENT',
campaignId: this.campaign._id });
    const createdPlacementsMap = new Map(createdPlacements.map(p =>
[p.name, p]));

    let { startDate, endDate } = dcmCampaign;

    await Promise.all(
      Object.keys(placements).map(async key => {
        const placement = placements[key];
        const createdPlacement = createdPlacementsMap.get(key);
        if (createdPlacement) {
          if (moment(startDate).unix() >
moment(this.campaign.dateStart).unix()) {
            await dcm.patchPlacement(createdPlacement.dcmId, {
              testingStartDate: startDate,
              pricingSchedule: {
                startDate,
                endDate,
                pricingPeriods: [{ startDate, endDate }],
              },
            });
          }
          return (placement.dcmId = createdPlacement.dcmId);
        }

        const result = await dcm.createPlacement({
          size: placement.params.size,
          siteId: placement.params.siteId,
          name: key,
          campaignId: this.campaign.dcmCampaignId,
          tagFormats: ['PLACEMENT_TAG_IFRAME_JAVASCRIPT'],
          paymentSource: 'PLACEMENT_AGENCY_PAID',
          compatibility: 'DISPLAY',
          pricingSchedule: {
            startDate: moment(this.campaign.dateStart).isAfter(startDate,
'day')
              ? moment(this.campaign.dateStart).format('YYYY-MM-DD')
              : startDate,
            endDate: moment(this.campaign.dateFinish).isBefore(endDate)
              ? moment(this.campaign.dateFinish).format('YYYY-MM-DD')
              : endDate,
            pricingType: 'PRICING_TYPE_CPM',
          },
        });
        const dcmId = result.data.id;
        const pl = new GoogleDCM({ name: key, type: 'PLACEMENT', dcmId,
campaignId: this.campaign._id });
```

```
        await pl.save();

        placement.dcmId = dcmId;
      }),
    );

    Object.keys(adGroups).forEach(key => {
      const { hash } = adGroups[key].params;
      adGroups[key].placements = Object.keys(placements).reduce((acc, key2)
=> {
        if (placements[key2].params.hash === hash)
acc.push(placements[key2]);
        return acc;
      }, []);
    });
    const variationsForCSV = [];
    const preparedVariations = {};

    const { creatives, excludeCreatives } = await
this.getCreativesForPairing(dcm);

    const createdAds = await GoogleDCM.find({ type: 'AD', campaignId:
this.campaign._id });
    const createdAdsMap = new Map(createdAds.map(a => [a.name, a]));

    await Promise.all(
      Object.keys(adGroups).map(async adKey => {
        const ad = adGroups[adKey];
        const formats = ad.placements.map(p => p.params.format);

        const technologyTargeting = ad.params.targeting.reduce((acc,
targeting) => {
          if (targeting.type === 'Language') return acc;
          const type = adTargetingsMap.get(targeting.type);
          if (type) {
            acc[type] = targeting.value.map(({ value }) =>
JSON.parse(value));
          }

          return acc;
        }, {});

        const languageTargeting = ad.params.targeting.reduce((acc,
targeting) => {
          if (targeting.type === 'Language') {
            const type = adTargetingsMap.get(targeting.type);
            if (type) {
              acc[type] = targeting.value.map(({ value }) =>
JSON.parse(value));
            }
          }
          return acc;
        }, {});

        let adId = null;
        const createdAd = createdAdsMap.get(adKey);

        const advert = {
          name: adKey,
          campaignId: this.campaign.dcmCampaignId,
          deliverySchedule: {
            frequencyCap: null,
            hardCutoff: false,
            impressionRatio: 1,
```

```
            priority: 'AD_PRIORITY_15',
          },
          placementAssignments: ad.placements.map(({ dcmId }) => ({ active:
true, placementId: dcmId })),
          technologyTargeting,
          languageTargeting,
          startTime:
            moment(this.campaign.dateStart).unix() >=
moment(startDate).unix() &&
            moment(this.campaign.dateStart).unix() < moment(endDate).unix()
              ? moment(this.campaign.dateStart).format()
              : moment(startDate).format(),
          endTime:
            moment(this.campaign.dateFinish).unix() <=
moment(endDate).unix() &&
            moment(this.campaign.dateFinish).unix() >
moment(startDate).unix()
              ? moment(this.campaign.dateFinish).format()
              : moment(endDate).format(),
        };

        await this.applyCreatives(dcm, creatives, excludeCreatives, advert,
formats, createdAd, ad);

        if (createdAd) {
          ad.dcmId = createdAd.dcmId;
          await dcm.updateAd({
            id: createdAd.dcmId,
            ...advert,
          });
          adId = createdAd.dcmId;
          createdAd.time = { startTime: advert.startTime, endTime:
advert.endTime };
          //          createdAd.creatives = []; // use it for update later
          await createdAd.save();
        } else {
          if (moment().isAfter(advert.startTime)) {
            advert.startTime = moment().format();
          }
          const { data } = await dcm.createAd(advert);
          const adItem = new GoogleDCM({
            name: adKey,
            type: 'AD',
            dcmId: data.id,
            campaignId: this.campaign._id,
            time: { startTime: advert.startTime, endTime: advert.endTime },
          });
          await adItem.save();
          ad.dcmId = adId = data.id;
        }

        ad.variations.forEach(variation => {
          const targeting = this.getVariationTargeting(variation).filter(t
=> !isTimeOfTheDay(t.name));
          const targetingStrings = targeting.map(t => ({
            name: t.name,
            value: typeof t.value === 'object' ? t.value.map(v =>
v.label).join(',') : t.value,
          }));
          variation.data = [...variation.data, ...targetingStrings];

          const placements = ad.placements.map(({ dcmId }) =>
dcmId).join(',');
```

```
          if (!variation.assembly.audienceDefault) {
            variation.data.push({ name: 'CM Ad ID', value: adId });
            variation.data.push({ name: 'CM Placement ID', value:
placements });
          }
          variationsForCSV.push(variation);

          preparedVariations[variation.hash] = { placementId: placements,
adId };
        });
      }),
    );

    const adverts = Object.values(adGroups);
    for (let ad of createdAds) {
      const removeAd = adverts.find(a => a.dcmId == ad.dcmId);
      if (!removeAd) {
        await dcm.patchAd(ad.dcmId, { active: false });
      }
    }

    const uploadedCreatives = await GoogleDCM.find({ type: 'CREATIVE',
campaignId: this.campaign._id });
    const uploadedCreativesMap = new Map(uploadedCreatives.map(c =>
[c.name, c]));

    try {
      const templates = await Templates.find({ campaignId:
this.campaign._id });

      await Promise.all(
        templates.map(async template => {
          const [width, height] = template.size.split('x');
          const path = await downloadAndUnpackStuff({ campaignId:
this.campaign._id });
          const defaultAdCretiveName =
`DEFAULT_AD_CREATIVE_${template.size}`;
          const uploadedCreative =
uploadedCreativesMap.get(defaultAdCretiveName);
          if (!uploadedCreative) {
            const backupImageExtensions = ['jpg', 'png'];
            const backupImagePaths = backupImageExtensions.map(
              e => `${path}/templates/${template.size}-
${template.name}/backup.${e}`,
            );
            const backupImagesStatus = await
Promise.all(backupImagePaths.map(path => fse.exists(path)));
            const backupImageIndex = backupImagesStatus.indexOf(true);
            if (backupImageIndex === -1) {
              throw Error(`None from ${backupImagePaths} exists`);
            }

            const media = { body:
fs.createReadStream(backupImagePaths[backupImageIndex]) };
            const resource = {
              assetIdentifier: { name:
`backup.${backupImageExtensions[backupImageIndex]}`, type: 'HTML_IMAGE' },
            };
            const creativeAsset = (await dcm.insertAsset(resource,
media)).data;
            const defaultCreative = (await dcm.insertCreative({
              type: 'DISPLAY',
              active: true,
              advertiserId: dcmAdvertiserId,
```

```
              creativeAssets: [creativeAsset],
              name: defaultAdCretiveName,
              size: { width, height },
            })).data;
            await dcm.createCampaignCreativeAssociation({
              campaignId: this.campaign.dcmCampaignId,
              creativeId: defaultCreative.id,
            });
            const dcmCreativeItem = new GoogleDCM({
              type: 'CREATIVE',
              name: defaultAdCretiveName,
              campaignId: this.campaign._id,
              dcmId: defaultCreative.id,
            });
            await dcmCreativeItem.save();
          }
        }),
      );
    } catch (e) {
      log.error('Cannot publish creative:', e);
    }
    return { CSV: variationsForCSV, variations: preparedVariations };
  }

  get getFieldsForReporter() {
    return [
      'campaignId',
      'campaignName',
      'partnerId',
      'partnerName',
      'advertiserId',
      'advertiserName',
      'variationId',
      'variationHash',
      'audienceId',
      'audienceName',
      'isDefault',
      'format',
      'placementId',
      'adId',
    ];
  }

  async generateCSVForReporter(campaignId) {
    if (Object.values(this.preparedVariations).length) {
      return super.generateCSVForReporter(campaignId,
this.preparedVariations);
    }
    return super.generateCSVForReporter(campaignId);
  }
}

module.exports = DCMTraffickedGooglePublisher;
```

facebook/dcmTracking.js

```
const moment = require('moment');
const GoogleDCMService = require('core/src/services/GoogleDCMService');
const { models } = require('db');
const log = require('../../../utils/logger');

const { Advertisers, GoogleDCM } = models;

const trackingSize = {
```

```
  width: 1,
  height: 1,
};

async function assignCreativeToAd(dcm, { adId, creativeId, clickThroughUrl
}) {
  try {
    const newData = {
      creativeRotation: {
        creativeAssignments: [],
      },
      active: true,
    };
    newData.creativeRotation.creativeAssignments.push({
      creativeId,
      clickThroughUrl: {
        customClickThroughUrl: clickThroughUrl,
      },
      active: true,
    });
    const result = await dcm.patchAd(adId, newData);
    return result;
  } catch (e) {
    throw Error('ASSIGN_CREATIVE_ERROR');
  }
}

module.exports = async function patchVariationsWithDcmTracking(campaign,
variations) {
  const { dcmAdvertiserId, dcmNetworkId } = await
Advertisers.findById(campaign.advertiserId);
  const dcm = new GoogleDCMService({ profileId: dcmNetworkId,
dcmAdvertiserId });
  const dcmCampaignId = campaign.facebookDcmCampaignId;
  const pendingPlacements = [];
  const pendingAds = [];
  const pendingCreativeAssignments = [];

  for (let i = 0, { length } = variations; i < length; i++) {
    const variation = variations[i];
    const { name: assemblyName } = variation.assembly;
    const placementName = `${assemblyName}-${variation.hash}`;

    const pendingPlacement = dcm
      .createPlacement({
        name: placementName,
        size: trackingSize,
        campaignId: dcmCampaignId,
        siteId: campaign.facebookDcmSites[0],
        tagFormats: ['PLACEMENT_TAG_IFRAME_JAVASCRIPT'], // TODO: check
        paymentSource: 'PLACEMENT_AGENCY_PAID',
        compatibility: 'DISPLAY',
        pricingSchedule: {
          startDate: moment(campaign.dateStart).format('YYYY-MM-DD'),
          endDate: moment(campaign.dateFinish).format('YYYY-MM-DD'),
          pricingType: 'PRICING_TYPE_CPM',
        },
      })
      .then(response => response.data);

    pendingPlacements.push(pendingPlacement);
  }

  const createdPlacements = await Promise.all(pendingPlacements);
```

```
  for (let i = 0, { length } = createdPlacements; i < length; i++) {
    const placement = createdPlacements[i];

    const advert = {
      name: placement.name,
      campaignId: dcmCampaignId,
      deliverySchedule: {
        frequencyCap: null,
        hardCutoff: false,
        impressionRatio: 1,
        priority: 'AD_PRIORITY_15',
      },
      type: 'AD_SERVING_TRACKING',
      startTime: moment(campaign.dateStart).format(),
      endTime: moment(campaign.dateFinish).format(),
      placementAssignments: [{ active: true, placementId: placement.id }],
    };
    const ad = dcm.createAd(advert).then(response => response.data);
    pendingAds.push(ad);
  }

  const ads = await Promise.all(pendingAds);
  const { data: createdCreative } = await dcm.insertCreative({
    type: 'TRACKING_TEXT',
    active: true,
    name: 'TRACKING CREATIVE',
    advertiserId: dcmAdvertiserId,
  });
  await dcm.linkCreativeToCampaign(createdCreative.id, dcmCampaignId);

  for (let i = 0, { length } = ads; i < length; i++) {
    const ad = ads[i];
    const variation = variations[0];
    const clickThroughUrl = variation.data.find(el => el.name === 'Website
URL').value;
    const pendingAssignment = assignCreativeToAd(dcm, { adId: ad.id,
creativeId: createdCreative.id, clickThroughUrl });
    pendingCreativeAssignments.push(pendingAssignment);
  }

  await Promise.all(pendingCreativeAssignments);
  const { placementTags } = await dcm
    .generatePlacementTags({
      campaignId: dcmCampaignId,
      placementIds: createdPlacements.map(pl => pl.id),
      tagFormats: ['placement_tag_tracking']
    })
    .then(response => response.data);

  const clickUrlMaps = new Map(placementTags.map(pl => [pl.placementId,
pl.tagDatas[0].clickTag]))

  const patchedVariations = variations.map((variation, index) => {
    const currPlacement = createdPlacements[index];
    const clickUrl = clickUrlMaps.get(currPlacement.id);
    const urlDataObject = variation.data.find(el => el.name === 'Website
URL');
    urlDataObject.value = clickUrl;
    return variation;
  });
  log.info(JSON.stringify(patchedVariations, null, 2));
  return patchedVariations;
};
```

facebook/index.js

```
const get = require('lodash/get');
const core = require('core');

const { Advertisers, Assemblies } = require('db').models;
const { facebookFieldTypes, fieldTypes, ruleTypes, channels, subChannels,
s3path } = require('common');

const VendorBase = require('../vendorBase');
const queueFacebookPublisher = require('./queue');
const publishFactory = require('./publishFactory');

const log = require('../../utils/logger');

const patchVariationsWithDcmTracking = require('./dcmTracking');

const { FacebookService } = core.facebook;
const { getAllLatestVersionVariations } = core.dals.variation;

const token = process.env.FACEBOOK_TOKEN;
const AssetUrlConfig = {
  S3_REGION: process.env.S3_REGION,
  S3_UPLOAD_BUCKET: process.env.S3_UPLOAD_BUCKET,
};

/**
 * publish
 *
 * @param {Campaign} campaign
 */
async function publish(campaign) {
  const campaignId = campaign._id;

  const variations = await getAllLatestVersionVariations(campaignId,
channels.facebook);

  log.info(`Publish Facebook: campaignId ${campaignId}, variations to
publish length ${variations.length}`);

  if (!variations.length) {
    log.info(`Publish Facebook finished. No variations found. campaignId
${campaignId}`);
    return;
  }

  // prepare variations for publishing
  const vars = variations.map(variation => {
    // prepare image url
    const data = variation.data.map(item => {
      let { value } = item;
      if (facebookFieldTypes[item.name] === fieldTypes.image) {
        const size = variation.template ? variation.template.size : null;
        const templateName = variation.template ? variation.template.name :
variation.assembly.format;

        const assetUrl = s3path.getAssetsUrl(
          AssetUrlConfig.S3_REGION,
          AssetUrlConfig.S3_UPLOAD_BUCKET,
          campaignId,
          size,
          templateName,
```

```
        );
        value = `${assetUrl}${item.value}`;
      }

      return { ...item, value };
    });

    return { ...variation, data };
  });
  let preparedVariations = [];
  const [facebookSubChannel] = campaign.facebookChannel;
  if (facebookSubChannel === subChannels.facebook.FACEBOOK_DCM_TRACKING)
    preparedVariations = await patchVariationsWithDcmTracking(campaign,
vars);
  else if (facebookSubChannel === subChannels.facebook.FACEBOOK_DIRECT)
preparedVariations = vars;
  else throw Error('UNSUPPORTED_FACEBOOK_SUBCHANNEL');

  // get advertiser params
  const paramsAdAccount = {
    facebookAdAccountId: null,
    facebookPageId: null,
  };

  const advertiser = await Advertisers.findById(campaign.advertiserId);
  paramsAdAccount.facebookAdAccountId = advertiser.facebookAdAccountId;

  // 1. create map adset->variationIds
  // 2. clear adsets
  //  - get all variations from adset
  //  - disable those that not exist in newly publish list
  //  - enable those that are enabled
  //  - remove from newly publish those with the same Ids
  // 3. publish variation list

  const adsets = preparedVariations.reduce((acc, variation) => {
    const adsetTargeting = variation.targeting.find(targeting =>
targeting.type === ruleTypes.facebook.adSetName);
    if (!adsetTargeting) return acc;
    const adsetId = adsetTargeting.value && adsetTargeting.value.value;
    if (!adsetId) return acc;

    if (!acc[adsetId]) {
      acc[adsetId] = [];
    }

    acc[adsetId].push(variation.hash);
    return acc;
  }, {});

  const fb = new FacebookService(token);

  for (const adsetId of Object.keys(adsets)) {
    // eslint-disable-next-line no-await-in-loop
    const ads = (await fb.getAds(adsetId)).map(ad => ({
      variationHash: ad.name.split('-')[1],
      name: ad.name,
    }));

    const variationHashesToDisable = ads
      .filter(
        ad =>
          !ad.name.startsWith(FacebookService.getDeletedPrefix()) &&
```

```
          !preparedVariations.find(variation => variation.hash ===
ad.variationHash),
      )
      .map(ad => ad.variationHash);

    log.info(
      `Disabling ads. ${variationHashesToDisable.length} ads to disable.
campaignId ${campaignId} adset ${adsetId}`,
    );

    // eslint-disable-next-line no-await-in-loop
    await fb.disableAds(adsetId, variationHashesToDisable);

    // find archived
    const variationHashesToEnable = ads
      .filter(
        ad =>
          ad.name.startsWith(FacebookService.getDeletedPrefix()) &&
          preparedVariations.find(variation => variation.hash ===
ad.variationHash),
      )
      .map(ad => ad.variationHash);

    log.info(
      `Enabling ads. ${variationHashesToEnable.length} ads to enable.
campaignId ${campaignId} adset ${adsetId}`,
    );

    // eslint-disable-next-line no-await-in-loop
    await fb.enableAds(adsetId, variationHashesToEnable);

    // remove those already exist
    preparedVariations = preparedVariations.filter(variation =>
!ads.find(ad => ad.variationHash === variation.hash));
  }

  log.info(`Sending to facebook publish. ${preparedVariations.length}
variations. campaignId ${campaignId}`);
  for (const variation of preparedVariations) {
    const fbPageIdTargeting = variation.targeting.find(targeting =>
targeting.type === ruleTypes.facebook.pageId);
    const fbPageId = get(fbPageIdTargeting, 'value.value');

    if (fbPageId) {
      paramsAdAccount.facebookPageId = fbPageId;
      queueFacebookPublisher.publish({ variation, paramsAdAccount });
    } else {
      log.info(`Error in FB targeting: no fbPageId defined. variationId
${variation._id} `);
    }
  }

  log.info(`Publish Facebook finished: campaignId ${campaignId}`);
}

async function publishFacebookVariation({ variation, paramsAdAccount }) {
  log.info(
    `Publish facebook variation. variationId: ${variation._id};
variationHash: ${
      variation.hash
    }; facebookAdAccountId: ${paramsAdAccount.facebookAdAccountId}`,
  );
```

```
  const assembly = await Assemblies.findOne({ _id: variation.assemblyId },
null, {
    lean: true,
  });
  if (!assembly) {
    log.info(`Assembly not found for variationHash: ${variation.hash}`);
  }

  const adsetTarget = variation.targeting.find(target => target.type ===
ruleTypes.facebook.adSetName);
  // {label, value}  where value store adsetId
  const adset = adsetTarget.value;
  const adsetId = adset.value;

  const publishMethod = publishFactory(variation, paramsAdAccount);
  await publishMethod.createCreative();

  const adName = `${assembly.name}-${variation.hash}`;

  await publishMethod.createAd(adName, adsetId);

  log.info(`Variation ${adName} published successfully`);
}

class VendorFacebook extends VendorBase {
  async publish() {
    await publish(this.campaign);
  }
}

module.exports = {
  publish,
  publishFacebookVariation,
  VendorFacebook,
};
```

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

"ЗАТВЕРДЖЕНО"

Науковий керівник кафедри

_____ І.А. Дичка

"___" __ _____ 2018 р.

ПРОГРАМНА ПЛАТФОРМА ДЛЯ АВТОМАТИЗАЦІЇ НАДАННЯ ПОСЛУГ РЕКЛАМНИМ КОМПАНІЯМ

Програма та методика тестування

ДП.045440-04-51

"ПОГОДЖЕНО"

Керівник проекту:

_____ Р.А. Гадиняк

Нормоконтроль:

_____ М.В. Онай

Виконавець:

___ _____П.О. Кривенко

2018

ЗМІСТ

1. ОБ’ЄКТ ВИПРОБУВАНЬ

Програмна платформа для автоматизації надання послуг рекламним компаніям, написаний на програмній платформі Node.js з використанням фреймворку Express та бібліотеки для клієнтської частини React.js.

2. МЕТА ТЕСТУВАННЯ

У процесі тестування має бути перевірено наступне:

1. Функціональна працездатність елементів сторінок користувацького інтерфейсу.
2. Взаємодію мікросервісів у системі.
3. Відмовостійкість черги повідомлень
4. Інтеграцію з медіа-майданчиками
5. Збір та агрегація статистики.
6. Забезпечення коректної обробки запитів від користувача.
7. Забезпечення належного рівня безпеки даних.
8. Зручність роботи з платформою.
9. Відповідність дизайну вимогам Технічного завдання.

3. МЕТОДИ ТЕСТУВАННЯ

Тестування виконується методом Gray Box Testing. Перевіряється як код, так і безпосередньо програмний продукт на відповідність функціональним вимогам. Тестування відбувається на рівні «системного тестування».

Використовуються наступні методи:

1. Функціональне тестування, зокрема на рівні Critical path test (базове тестування).
2. Тестування продуктивності програмного забезпечення, зокрема Performance testing (тестування стабільності).
3. Тестування інтерфейсу.

4. ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Працездатність web-додатку перевіряється шляхом:

1. Динамічного ручного тестування – введенням граничних та недопустимих значень в поля, які можна редагувати.
2. Динамічного ручного тестування на відповідність функціональним вимогам.
3. Статичного тестування коду.
4. Тестування web-ресурсу в різних web-браузерах.
5. Тестування при максимальному навантаженні.
6. Тестування стабільності роботи при різних умовах.
7. Тестування зручності використання.
8. Тестування інтерфейсу.

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

"ЗАТВЕРДЖЕНО"

Науковий керівник кафедри

_____ І.А. Дичка

"___" _____ 2019 р.

ПРОГРАМНА ПЛАТФОРМА ДЛЯ АВТОМАТИЗАЦІЇ НАДАННЯ ПОСЛУГ РЕКЛАМНИМ КОМПАНІЯМ

Керівництво користувача

ДП.045440-05-34

"ПОГОДЖЕНО"

Керівник проекту:

_____ Р.А. Гадиняк

Нормоконтроль:

_____ М.В. Онай

Виконавець:

_____ П.О. Кривенко

2019

ЗМІСТ

1. Опис структури програмних засобів

Програмні засоби включають в себе web-додаток, що складається із статичних сторінок, що віддаються сервером вже у остаточному вигляді і контент яких не змінюється, а також динамічних сторінок, дані яких регулярно оновлюються. Користувацький інтерфейс додатку є англомовним.

До статичних сторінок належать:

- сторінка авторизації;
- сторінка створення нового бренду;
- сторінка створення рекламного акаунту;
- сторінка, що повідомляє про технічні роботи на сервері.

До динамічних сторінок web-додатку належать:

- список користувачів;
- сторінка створення користувача;
- сторінка перегляду та редагування користувача;
- сторінка створення брифу;
- список створених брифів;
- сторінка створення нової кампанії;
- сторінка перегляду та редагування кампанії;
- сторінка перегляду та редагування шаблонів кампанії;
- сторінка зі списком завантажених шаблонів кампанії;
- сторінка перегляду та редагування активів кампанії;
- сторінка зі списком завантажених активів кампанії;
- сторінка перегляду варіацій кампанії;
- сторінка перегляду коментарів для варіацій;
- сторінка підтвердження варіацій;
- сторінка для створення набору даних для варіацій;
- сторінка зі списком брендів;
- список рекламних акаунтів.

Кожна web-сторінка містить навігаційне меню, що розташоване у лівій частині сторінки а також заголовок, який контент якого адаптується під конкретну сторінку. У нижній частині навігаційного меню знаходиться кнопка для виходу з системи. У випадку адміністративного користувача над кнопкою виходу з системи знаходиться випадаюче меню у якому можна переключатися між кабінетами різних партнерів.

2. Опис вмісту статичних web-сторінок

Неавторизовані користувачі мають доступ тільки до сторінки аутентифікації, для усіх інших сторінок необхідно мати акаунт користувача у системі та авторизуватися за його даними.

Сторінка аутентифікації (рис. 1) містить поля для вводу логіну та паролю користувача, а також кнопку для продовження процесу аутентифікації.

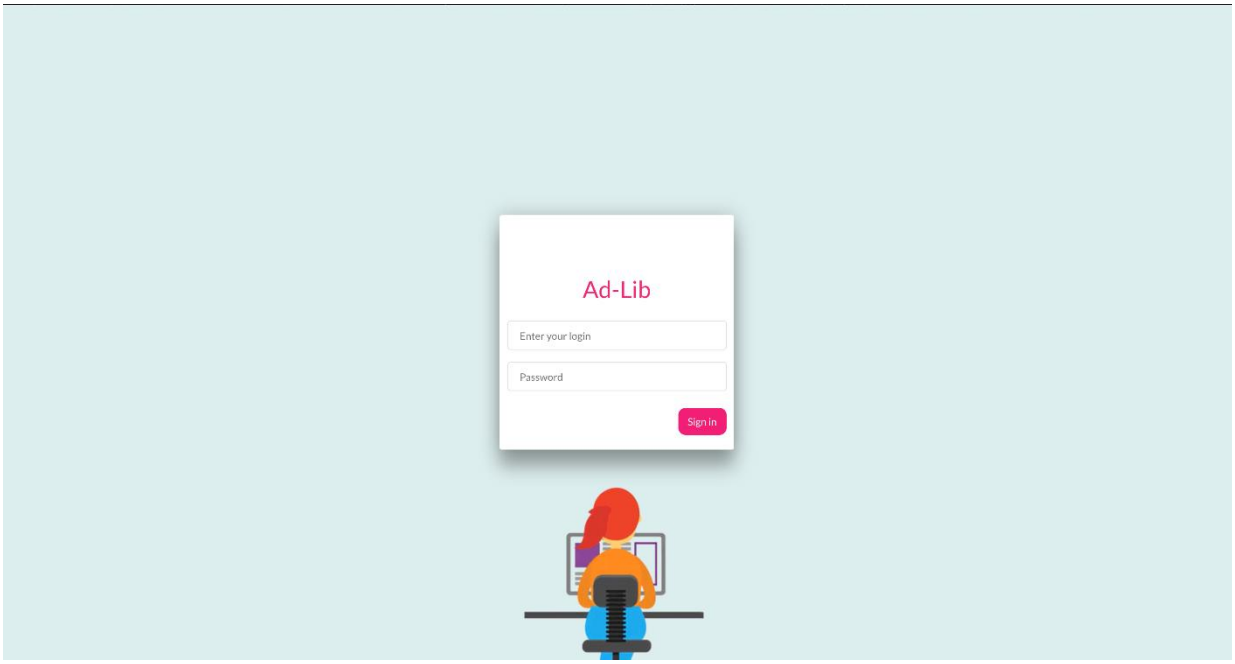


Рис. 1. Сторінка аутентифікації

Сторінка реєстрації користувачів містить поля для вводу логіну користувача, його імені, поле паролю, а також поле для підтвердження

паролю. Також форма містить випадаючий список користувацьких ролей. В залежності від вибору ролі у формі з'являться додаткові поля для заповнення. У обох формах поля для усі поля для паролю є прихованими, тобто текст, що вводиться до цих полей замасковано крапками.

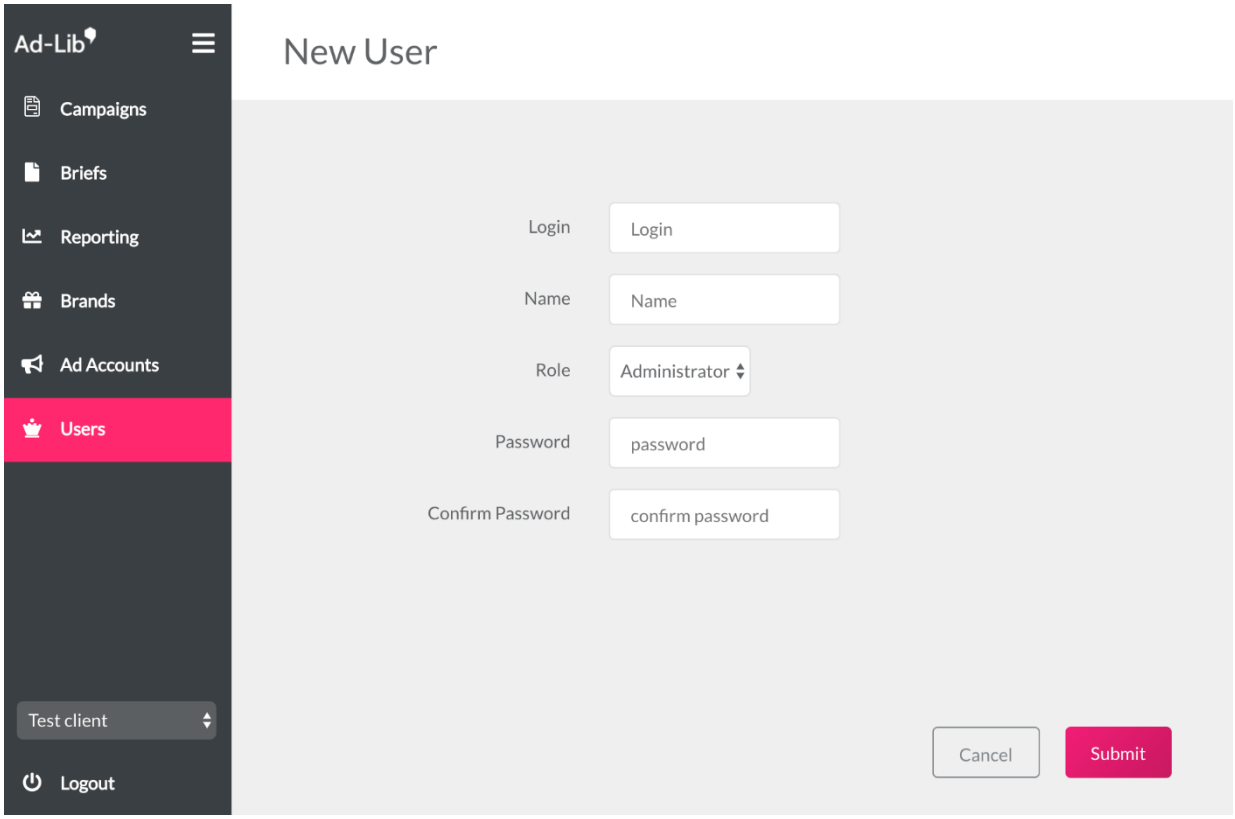


Рис. 2. Сторінка реєстрації

3. Процедура реєстрації користувача

Сторінка реєстрації користувача (рис. 2) доступна тільки адміністративним ролям, а саме адміністратору та головному адміністратору. Адміністратор має ввести логін, ім'я нового користувача та пароль для нового користувача, для того, щоб уникнути небажаних помилок адміністратор має повторити пароль у спеціальному для цього полі. В залежності від обраної ролі нового користувача адміністратору будуть запропоновані додаткові поля для заповнення. У випадку створення кабінету для партнера чи бренду необхідно вибрати партнера зі списку для якого

створюється даний акаунт. У випадку створення кабінету для рекламодавця також необхідно вибрати рекламний акаунт даного користувача. Після цього треба натиснути кнопку «Submit», що призведе до реєстрації користувача у системі.

4. Процедура авторизації користувача

Аутентифікація користувача відбувається на сторінці аутентифікації. Користувач має ввести свій логін та пароль. Після чого йому треба натиснути кнопку «Sign in», що призведе до авторизації користувача у системі. При неправильно введеному логіну чи паролю, користувач не зможе авторизуватися у системі, та побачить повідомлення про помилку.

5. Перегляд створених користувачів

Для перегляду користувачів, що були створені поточним адміністратором, необхідно обрати у лівому навігаційному меню пункт «Users», що відкриє сторінку користувачів (рис. 3). Кожен елемент зі списку користувачів є активним посиланням на сторінку перегляду та редагування обраного користувача.

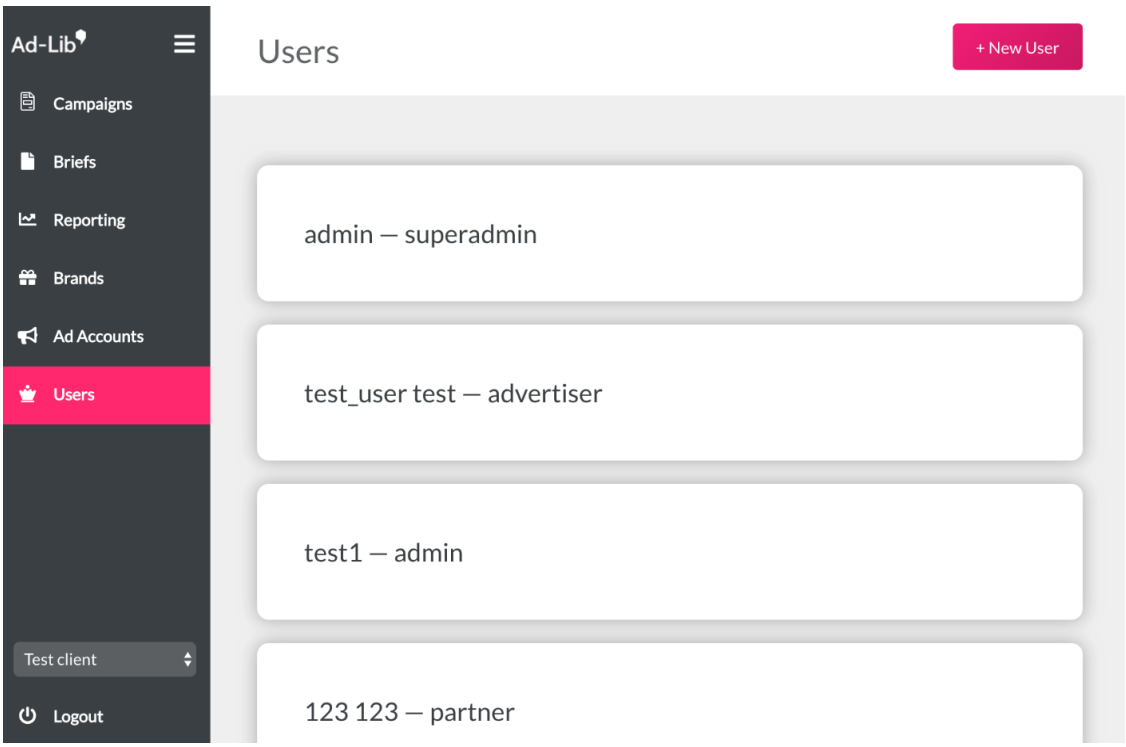


Рис. 3. Сторінка перегляду списку користувачів

6. Створення нової рекламної кампанії

Для створення нової рекламної кампанії необхідно обрати у лівому навігаційному меню пункт «Campaigns» та натиснути кнопку «New campaign». Користувачу пропонується заповнити наступні поля: назва нової кампанії, дати початку та кінцю, рекламний акаунт, та бриф на основі якого була створена кампанія. Також необхідно обрати канал для публікації та підтип публікації для обраного каналу. В залежності від обраного каналу та підтипу необхідно заповнити інформацію, що є обов'язковою для взаємодії з обраним каналом. Це може бути конвенція імен, рекламна кампанія, що створена у обраному каналі публікації ті ін. Після заповнення усіх полів потрібно натиснути на кнопку «Create», після чого користувача перенаправить на сторінку перегляду та редагування щойно створеної кампанії. Якщо певні необхідні поля не були заповненими на момент підтвердження створення нової кампанії, то ці поля підсвічуються рожевим кольором.

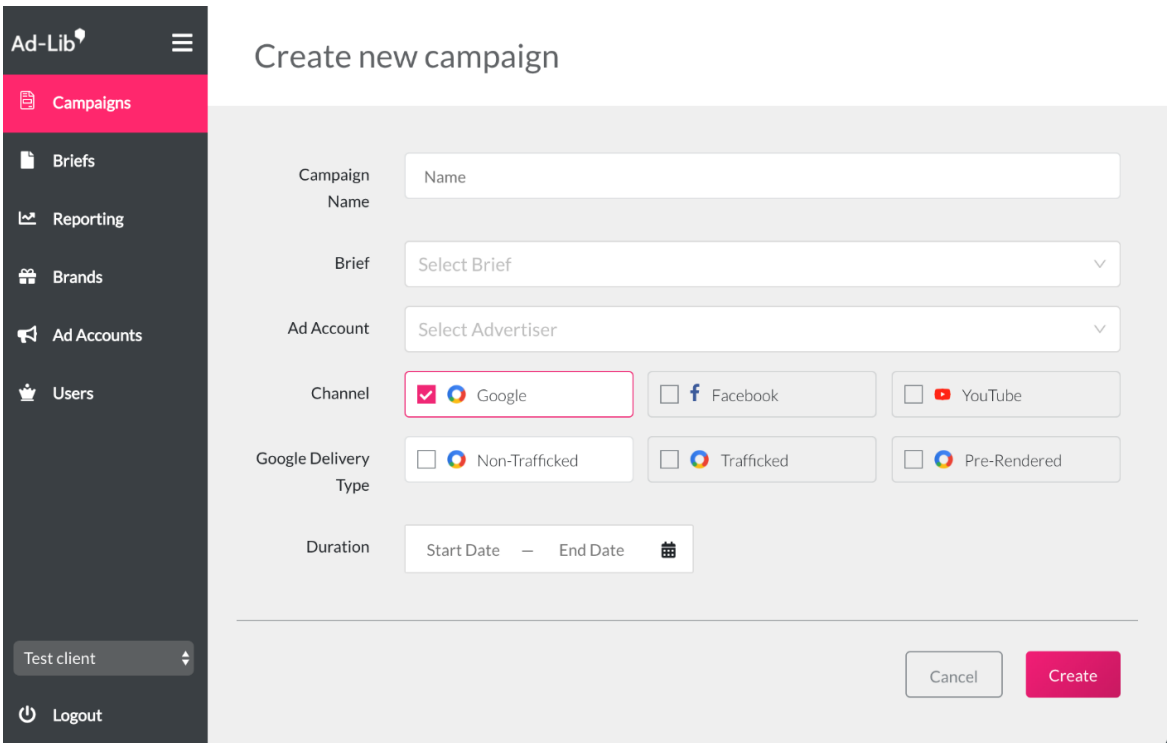


Рис. 4. Сторінка створення нової кампанії

7. Перегляд списку створених рекламних кампаній

Для перегляду кампаній, що були створені поточним адміністратором, необхідно обрати у лівому навігаційному меню пункт «Campaigns», що відкриє сторінку кампаній (рис. 5). Кожен елемент зі списку кампаній є активним посиланням на сторінку перегляду та редагування обраної кампанії.

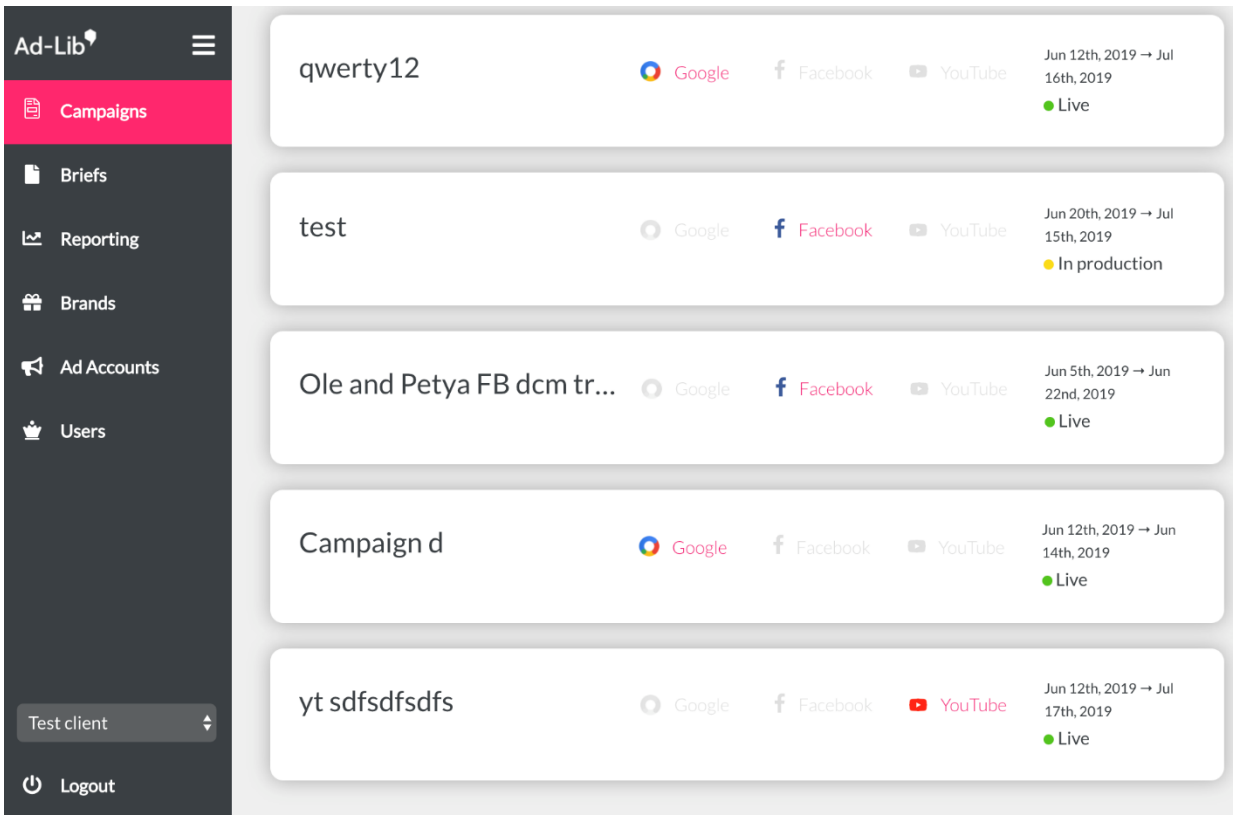


Рис. 5. Сторінка списку кампаній

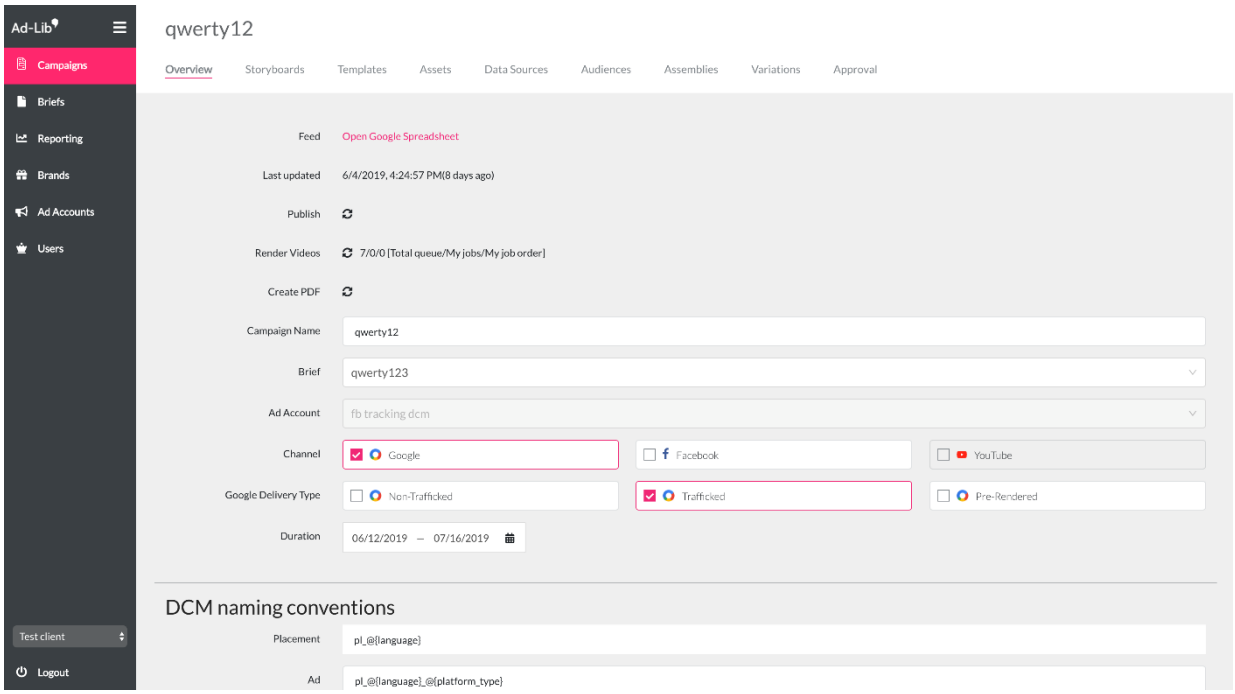


Рис. 6. Сторінка перегляду основної інформації кампанії

8. Перегляд рекламної кампанії

Обравши в наведеному списку кампанію, користувач потрапляє на сторінку цієї кампанії, на першу її вкладку «Overview»(рис. 6). На цій вкладці міститься така інформація як назва кампанії, обрані канали для публікації креативів, дати початку та кінця, обраний рекламний акаунт та ін.

Окрім вкладки «Overview», на сторінці перегляду кампанії знаходяться такі вкладки «Templates», «Assets», «Data Sources», «Audiences», «Assemblies», «Variations» та «Approval»(рис. 6).

На вкладці «Templates» знаходиться поле для завантаження нового шаблону для поточної рекламної кампанії, а також список шаблонів які були завантажені (рис. 7). Шаблон необхідно завантажити у форматі zip. У ньому має бути папка у імені якої вказано формат шаблону, а також назву кампанії для якої він завантажується. Кожен елемент списку містить інформацію, щодо формату поточного шаблону, його назву та список знайдених у ньому динамічних полів. Крім того, у правій частині картки знаходиться кнопка для перегляду та видалення цього шаблону.

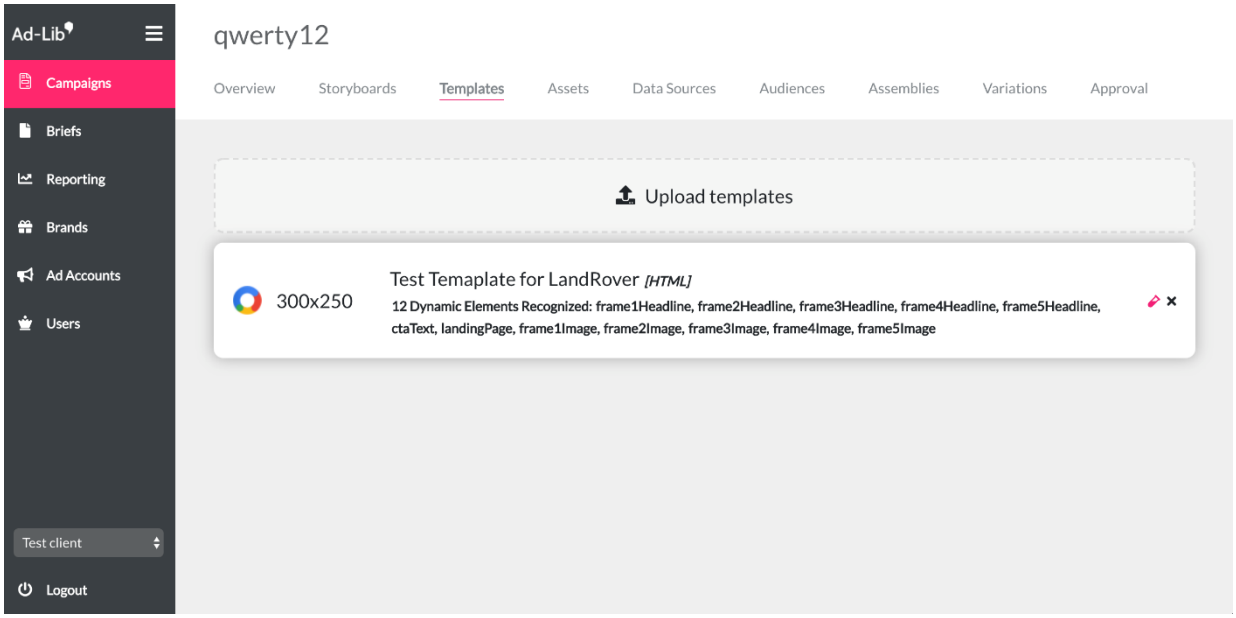


Рис. 7. Сторінка керування шаблонами кампанії

На вкладці Assets у верхній частині сторінки розташоване поле на всю ширину сторінки для завантаження цифрових активів, таких як зображення та відео. Під цим полем знаходиться список із вже завантажених активів. На кожному елементі цього списку знаходиться інформація, щодо того, які формати у ньому присутні, а також перелік знайдених файлів. У правій частині кожного елементу зі списку знаходиться кнопка для видалення даного активу з кампанії.

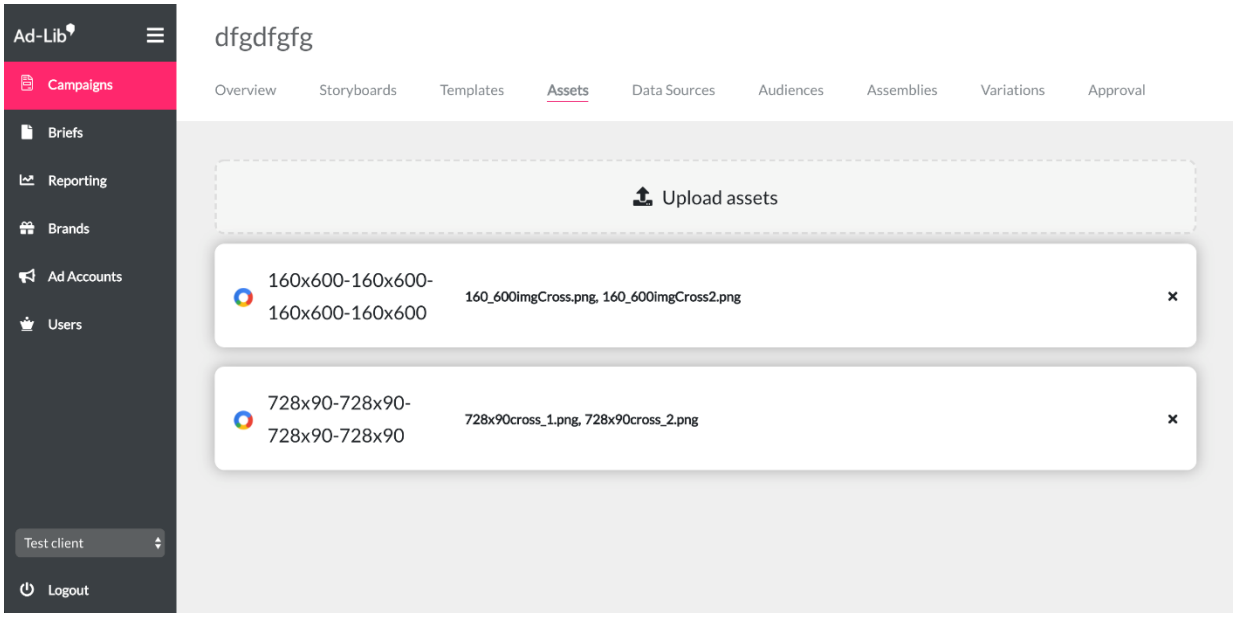


Рис. 8. Сторінка керування медіа-активами

На вкладці Data Sources також знаходиться поле на всю ширину сторінки для завантаження файлів з даними. Під ним розташований список з карток, що містить інформацію щодо кожного завантаженого файлу: назва, перелік знайдених колонок з даними та кнопка для видалення елементів, що були завантажені.

9. Створення даних для генерації варіацій

Щоб створити набір даних з яких згенеруються варіації необхідно перейти на вкладку «Assemblies» бажаної кампанії. Спочатку потрібно

задати унікальне ім’я для цього набору в рамках поточної кампанії. Далі потрібно задати динамічні дані для наявних полів, що були знайдені у завантаженому шаблоні. Дані можна вносити вручну до цих текстових полів, або використовувати csv файли як джерела даних. Для задання динамічних зображень необхідно завантажити архів з зображеннями на вкладці «Assets». Аналогічно, щоб мати можливість використовувати csv файли для динамічних полів спершу необхідно завантажити бажаний файл до поточної кампанії на вкладці «Data Sources». Після внесення всіх необхідних даних, на основі яких генеруються варіації потрібно задати ім’я для цієї збірки та зберегти її. На основі створеного набору даних згенерується набір варіацій. Для їх перегляду та підтвердження потрібно перейти до вкладки «Approval».

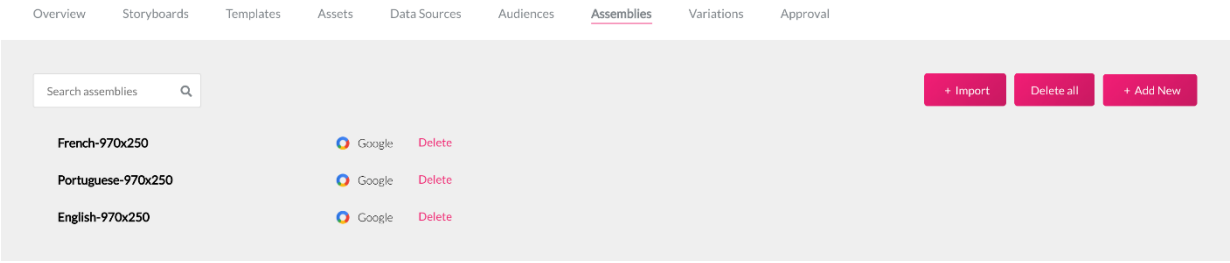


Рис. 9. Сторінка з наборами даних для варіацій

10. Перегляд та підтвердження згенерованих варіацій

На вкладці «Approval» знаходяться згенеровані варіації, які можна переглянути разом з шаблоном. Список варіацій можна фільтрувати за статусом, наявністю коментарів, тощо. Для кожної варіації можна окремо задати її статус «Approved» чи «Rejected», залишити коментар, а також переглянути вже існуючий, якщо такий наявний. У верхньому правому кутку сторінки знаходиться лічильник, що показує загальну кількість наявних варіацій, кількість варіацій, що були підтверджені, а також число тих, що були заперечені.

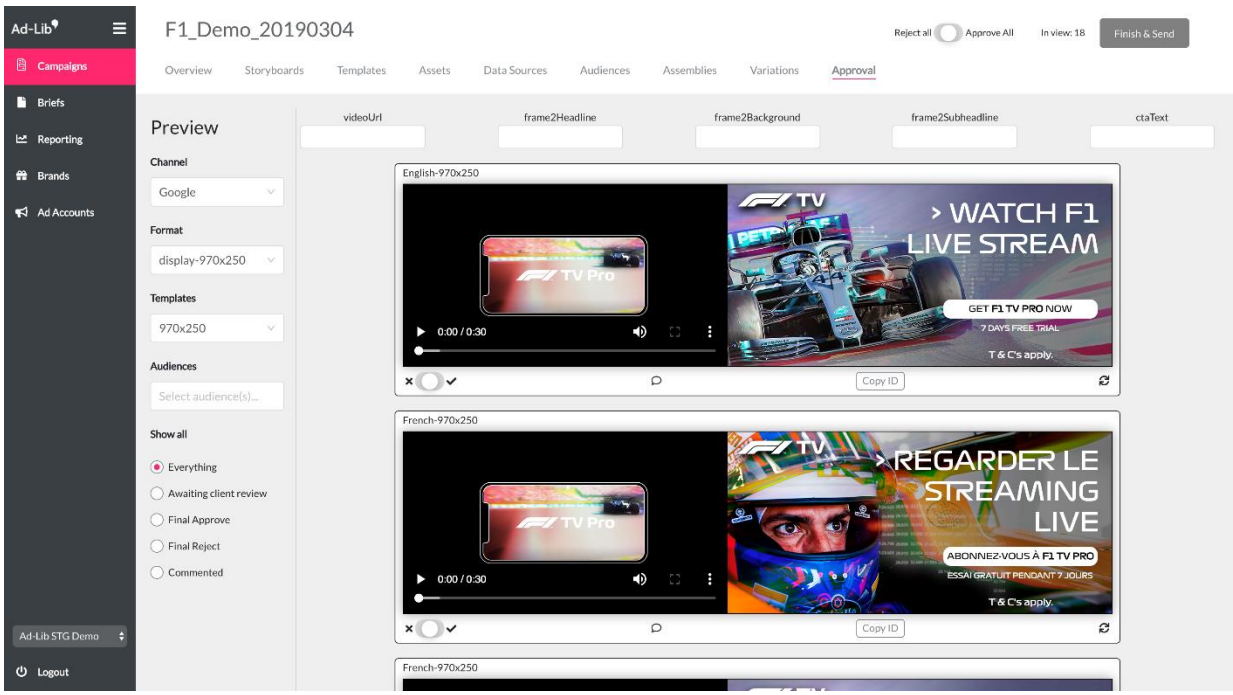


Рис. 10. Сторінка підтвердження варіацій

11. Публікація варіацій до медіа-майданчиків

Після підтвердження конкретних варіацій до публікації, потрібно повернутися на вкладку «Overview», де над полем з іменем кампанії знаходиться кнопка «Publish», яку необхідно натиснути для відправки варіацій на публікацію до обраного медіа-майданчика.